DOCUMENT RESUME

ED 194 060		IR 008 912

AUTHOR	Harris, Diana, Ed.; Collison, Beth, Ed.
TITLE	Proceedings of NECC/2 National Educational Computing Conference 1980 (Norfolk, Virginia, June 23-25, 1980).
INSTITUTION	Iowa Univ., Iowa City. Computer Center.
REPORT NO	ISBN-0-937114-00-6
PUB DATE	Jun 80
NOTE	306p.
EDRS PRICE	MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS	*Computer Assisted Instruction; Computer Assisted Testing; Computer Oriented Programs; Computer Programs; *Computers; *Computer Science; Elementary Secondary Education; Games; Higher Education; Humanistic Education; Learning Laboratories; Material Development; Mathematics Instruction; *Microcomputers; Science Education
IDENTIFIERS	*Computer Literacy

ABSTRACT

This proceedings, which includes 52 papers and abstracts of 13 invited and nine tutorial sessions, provides an overview of the current status of computer usage in education and offers substantive forecasts for academic computing. Papers are presented under the following headings: Business--Economics, Tools and Techniques for Instruction, Computers in Humanistic Studies, Computer Literacy, Science and Engineering, Structured Programming, ACM Elementary and Secondary Schools Subcommittee, Computer Science Education, Integrating Computing into K-12 Curriculum, Mathematics, Testing-Placement, Pre-College Instructional Materials, Minority Institutions--ECMI, Computer Laboratories in Education, Computer Games in Instruction, and Computing Curricula. Abstracts are provided for 13 invited sessions dealing with such topics as microcomputers in education; research in microcomputer uses; personal computing; educational computing: past, present, and future; the Open University; CAUSE program and projects; funding academic computing programs; computer-based resource sharing; computers and instruction; improving utilization of 2-year college computer centers; teaching computer ethics; data sets available from the federal government; and MIS education, as well as nine tutorial sessions designed to provide attendees with the opportunity to expand their appreciation of and involvement in educational computing. (CHC)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED194060

U S DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

Proceedings of NECC/2
National Educational Computing Conference 1980

Edited by
Diana Harris
Beth Collison

Hosted by
Christopher Newport College
Newport News, Virginia

Held at
Holiday Inn/Scope
Norfolk, Virginia

23, 24, 25 June 1980

"PERMISSION TO REPRODUCE THIS MATERIAL IN MICROFICHE ONLY HAS BEEN GRANTED BY
Ted Sjoerdama

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

IR008912

International Standard Book Number: 0-937114-00-6
Copyright 1980: NECC
Published by
The University of Iowa
Weeg Computing Center
Iowa City, Iowa 52242
for
National Educational Computing Conference II
June 1980

Cover design by Madeline Windauer

NECC 1980 Steering Committee

Alfred Bork
University of California-Irvine
Bobby Brown
University of Iowa
William S. Dorn
University of Denver
Karen Duncan
Gerald L. Engel
Christopher Newport College
Norman Gibbs
College of William and Mary
John W. Hamblen
University of Missouri-Rolla
Paul Hazen
Johns Hopkins University
Harry Hedges
Michigan State University
Lawrence A. Jehn
University of Dayton
Sister Mary Kenneth Keller
Clarke College
William E. Knabe
University of Iowa
David R. Kniefel
New Jersey Computer Network
Doris K. Lidtke
Towson State University
Elizabeth Little
EDUCOM/EDUNET
Sister Patricia Marshall
Xavier University of Louisiana
Richard E. Pogue
Medical College of Georgia
James Poirot
North Texas State University
Nancy Roberts
Lesley College
Theodore J. Sjoerdsma
University of Iowa
David L. Stonehill
University of Rochester
David B. Thomas
University of Iowa

General Chairman: Gerald Engel
Program Co-chairmen: Richard H. Austing, Doris K. Lidtke
Local Arrangements Committee: Robert Mathis, Michael Staman
Publications Committee: Diana Harris, Theodore Sjoerdsma, Madeline Windauer

The National Educational Computing Conference wishes to thank the following people for their contribution of effort, time, and knowledge as referees for the papers submitted for presentation.

Robert Aiken, University of Tennessee
William Atchison, University of Maryland
Emile Attala, Polytechnic State University
R. P. Banaugh, University of Montana
Bruce Barnes, National Science Foundation
Joyce Baucom, Southern University
Alfred Bork, University of California
G. R. Boynton, Indiana University
Mary Jane Brannon, Huntingdon College
Hans Brey, Tennessee Tech. University
Tom Carroll, Michigan State University
N. John Castellan, Indiana University
Sylvia Charp, School District of Philadelphia
Ronald Collins, Eastern Michigan University
Frank W. Connolly, Montgomery College
Michael J. D'Amore, New Jersey Educational Computing Network
Charles Davidson, University of Wisconsin
Herbert Dershem, Hope College
David Evans, VIMS
Selby Evans, Texas Christian University
Stefan Feyock, College of William and Mary
F. T. Fink, Michigan State University
Fred Gage, Texas Christian University
Norman Gibbs, College of William and Mary
Sheldon P. Gordon, Suffolk County Community College
Keith Hall, Ohio State University
John W. Hamblen, University of Missouri
Harry Hedges, Michigan State University
Darlene Heinrich, Florida State University
James A. Higgins, Digital Equipment Corp.
A. A. J. Hoffman, Texas Christian University
Lawrence Jehn, University of Dayton
Vincent H. Jones, Baton Rogue, Louisiana
Basheer Khumawala, University of North Carolina
Joyce C. Little, Community College of Baltimore
Richard W. Lott, Bentley College
David Maharry, Wabash College
Sister Patricia Marshall, Xavier University
Donald H. McClain, University of Iowa
Donald E. McLaughlin, Augustana College
Edward D. Meyers Jr., Center for the Study of Health Development
Percy L. Milligan, Southern University
Robert D. Montgomery, North Carolina Central University
Catherine Morgan, Kensington, Maryland
Mike Moshell, University of Tennessee
Robert Noonan, College of William and Mary
Bruce E. Norcron, SUNY at Binghamton
Linda Petty, Hampton Institute
Charles Pfleeger, University of Tennessee
Richard Pogue, Medical College of Georgia
W. W. Porterfield, Hampden-Sydney College
James Powell, Burroughs-Wellcome Co.
C. A. Quarles, Texas Christian University
Joe Raben, Queen College of SUNY
Ottis Rechard, University of Denver
David Rine, Western Illinois University
Peter Rizza, Control Data Corp.
Leroy Roquemore, Southern University
R. C. Rosenberg, Michigan State University
Ted Sjoerdsma, University of Iowa
David Smith, Duke University
Philip F. Spelt, Wabash College
Elliot A. Tanis, Hope College
Robert Tannenbaum, Ancram, New York
David Thomas, University of Iowa
Robert Thompson, University of Dayton
John Van Iwaarden, Hope College
Rita Wagstaff, Temple University
T. C. Willoughby, Ball State University
Gary Wittlich, Indiana University
Allen Ziebur, SUNY at Binghamton

And special thanks to the authors whose cooperation kept this publication on schedule. --Diana Harris

Foreword

The Second National Educational Computing Conference (NECC/2) builds on the success of last year's conference at the University of Iowa. In organizing the program for NECC/2 we have attempted to implement many of the suggestions we have received during the year. Of special significance are the tutorial sessions which are designed to provide attendees with the opportunity to expand their appreciation of and involvement in educational computing.

NECC/2 is a broadly based conference bringing together, in the common interests of computers in education, a great number of individuals with diverse backgrounds and a great number of cooperating societies. It is hoped that in this forum the diversity can be focused to improve our common interest.

This volume presents the papers presented at the conference and summaries of most of the special sessions. The coordination of the contributed papers was in the able hands of Richard H. Austing of the University of Maryland, and the coordination of the special sessions was in the equally able hands of Doris K. Lidtke of Towson State University. We give them our sincerest thanks for the many hours spent on these tasks. We also thank the authors for submitting their works and working with us to maintain our schedule.

Thanks are also due to the entire NECC/2 Steering Committee for their excellent guidance in preparing for the Conference.

Special acknowledgements go to Ted Sjoerdsma of the University of Iowa who coordinated publicity and Diana Harris also of the University of Iowa who edited these proceedings.

Finally we thank all those individuals who came to NECC/2 and helped insure that the concept of this series of conferences is a success.

Gerald L. Engel
Chairman, NECC/2
Christopher Newport College
Newport News, Virginia 23606

Table of Contents

TUTORIALS

1 CAI-An Introduction
Michael Arenson and Harold Rahmlow
2 How to Choose a Microcomputer for Educational Use
Kevin Hausmann
3 Turning Students on to the Computer: The Introductory Course
Gary Shelly
4 Instructional Design
Gary Stokes

INVITED SESSION

5 Microcomputers in Education
Chaired by Murali R. Varanasi

TUTORIAL

6 Program Development Techniques
A. J. Turner

BUSINESS/ECONOMICS

7 Computer Science and MIS College Students: An Investigation of Career-Related Characteristics
Eleanor W. Jordan
12 Force-Feeding SPSS in Market Research and Analysis at Hampton Institute
Howard F. Wehrle III
16 Short-Run Forecasting of the U.S. Economy
William R. Bowman

TOOLS AND TECHNIQUES FOR INSTRUCTION

19 Hypertext: A General-Purpose Educational Computer Tool
Darrell L. Ward and Steve Bush
25 A Dynamic Process in Teaching Techniques
Jamil E. Effarah
31 Considerations and Guidelines for Developing Basic Skills Curriculum for Use with Microcomputer Technology
Robert M. Caldwell

INVITED SESSION

37 Research in Microcomputer Uses in Education
Chaired by David Kniefel

COMPUTERS IN HUMANISTIC STUDIES

38 Creativity through the Microcomputer
George M. Bass, Jr.
42 Giving Advice with a Computer
James W. Garson
46 From a Theory of Reading to Practice via the Computer
Dale M. Johnson and R. Scott Baldwin
54 Non-Harmony: A Vital Element of Ear-Training in Music CAI
Joan C. Groom-Thornton and Antoinette Tracy Corbet

7

COMPUTER LITERACY

58 A Case for Information Literacy
Bruce B. Schimming
62 A Byte of BASIC
Judith A. Hopper
65 A Computer Workshop for Elementary and Secondary Teachers
Herbert L. Dershem and John T. Whittle
68 Microcomputers and Computer Literacy: A Case Study
Robert J. Ellison

INVITED SESSION

73 Personal Computing: An Adventure of the Mind
Paul Hazen

INVITED SESSIONS

74 Educational Computing: Past, Present, and Future
Ronald W. Collins
75 The Open University
Frank Lovis and William Dorn

SCIENCE AND ENGINEERING

76 Demographic Techniques in Ecology: Computer-Enhanced Learning
A. John Gatz, Jr.
81 Microcomputers as Laboratory Instruments: Two Applications in Neurobiology
Richard F. Olivo
Classical Mechanics with Computer Assistance
A. Douglas Davis
90 Computer-Augmented Video Education in Electrical Engineering at the U.S. Naval Academy
Tian S. Lim, Michael W. Hagee, and Richard A. Pollak

STRUCTURED PROGRAMMING

96 The Use of Programming Methodology in Introductory Computer Science Courses
Elizabeth Alpert
103 FORTRAN 77: Impact on Introductory Courses in Programming Using FORTRAN
Frank L. Friedman
112 Using Model-Based Instruction to Teach PASCAL
Bodgan Czejdo
119 Structured Machine-Language: An Introduction to Both Low- and High-Level Programming
David G. Hannay

ACM ELEMENTARY AND SECONDARY SCHOOLS SUBCOMMITTEE

125 ACM Elementary and Secondary Schools Subcommittee Progress Report
David Moursund
130 Computing Competencies for School Teachers
Robert P. Taylor, James L. Poirot, and James D. Powell

INVITED SESSION

137 CAUSE Program and Projects
Chaired by Lawrence Oliver

TUTORIAL

138 Databases - What Are They?
Arlan DeKock

COMPUTER SCIENCE EDUCATION

139 Required Freshman Computer Education in a Liberal Arts College
David E. Wetmore
143 Development of Communications Skills in Software Engineering
John A. Beidler and John G. Meinke
147 Systematic Assessment of Programming Assignments
Judy M. Bishop
152 Data Structures at the Associate Degree Level
Richard F. Dempsey

INTEGRATING COMPUTING INTO K-12 CURRICULUM

155 The Scarsdale Project: Integrating Computing into the K-12 Curriculm
Thomas Sobol and Robert P. Taylor
168 Panel
Beverly Hunter and Catherine E. Morgan

INVITED SESSION

169 Funding Academic Computing Programs
Sheldon P. Gordon and Lawrence Oliver

TUTORIAL

170 Techniques for Instructional Software Development Using Microcomputers
Kevin Hausmann

MATHEMATICS

171 A Method for Experimenting with Calculus Using CAI
Frank D. Anger and Rita V. Rodriguez
179 Computer Applications in a Finite Mathematics Course
G. Piegari, K. Abernethy, and A. L. Thorsen
184 A Computer-Assisted Course in Biomathematics
Pui-Kei Wong
194 Computer Symbolic Math
David R. Stoutemyer

INVITED SESSIONS

197 Computer-Based Resource Sharing
Donna Davis Mebane and Rodney Mebane
198 Computers and Instruction: Development, Directions, and Alternatives
Chaired by William Gruener

TUTORIAL

199 Videodisc
Bobby Brown and Joan Sustick

TESTING/PLACEMENT

200 Microcomputer-Assisted Study and Testing System
Hugh Garraway
205 RIBYT - A Database System for Formal Testing and Self-Assessment
F. Paul Fuhs
214 Computer-Managed Placement in Mathematics Instruction for Health Occupations Students
Thomas A. Boyle and Peter Magnant

INVITED SESSIONS

220 Improving Utilization of Two Year College Computer Centers
Chaired by Joyce Currie Little
221 Teaching Computer Ethics (Workshop)
Walter Maner

TUTORIAL

222 PASCAL
H. P. Haiduk

PRE-COLLEGE INSTRUCTIONAL MATERIALS

223 Computer-Based Instruction for the Public Schools: A Suitable Task for Microprocessors?
Timothy Taylor
230 Microcomputer/Videodisc CAI - Some Development Considerations
Ron Thorkildsen and Kim Allard
236 The Non-Technical Factors in the Development of CAI
Michael Mocciola

INVITED SESSIONS

237 Data Sets Available from the Federal Government
Chaired by Thomas E. Brown

MINORITY INSTITUTIONS - ECMI

238 Academic Computing: A Sampler of Approaches in Minority Institutions
Sr. Patricia Marshall
245 Computer Use in Chemistry at a Minority Institution
James D. Beck
249 Educational Use of Computers in Puerto Rico
Frank D. Anger

COMPUTER LABORATORIES IN EDUCATION

250 Microcomputers in the Teaching Lab
Robert F. Tinker
256 The Computer Lab of the 80s
Guy Larry Brown
258 The Education Technology Center
Alfred Bork, Stephen Franklin, and Barry Kurtz

INVITED SESSION

260 MIS Education: Industry Needs and Educational Solutions
Chaired by Eleanor W. Jordan

COMPUTER GAMES IN INSTRUCTION

261 Shall We Teach Structured Programming to Children?
Jacques E. LaFrance
266 Structured Gaming: Play and Work in High School Computer Science
J. M. Moshell, G. W. Amann, and W. E. Baird
271 Tapping the Appeal of Games in Instruction
Peter O. McVay

COMPUTING CURRICULA

276 An Educational Program in Medical Computing for Clinicians and Health Scientists
Albert Hybl and James A. Reggia
281 A Secondary Level Curriculum in System Dynamics
Nancy Roberts and Ralph M. Deal
287 The Computer Software Technician Program at Portland Community College
David M. Hata
290 A Computer Science Major in a Small Liberal Arts College
Joerg Mayer

293 Author Index

11

Tutorials

CAI - AN INTRODUCTION

Michael Arenson
Dept. of Music
Univ. of Delaware
Newark, Delaware 19711
(302) 738-8485
Harold Rahmlow
Abacus Learning Inc.
531 Lancaster Avenue
Wayne, Pennsylvania 19087
The American College
Bryn Mawr, Pennsylvania 19010

ABSTRACT

The NECC-2 Computer-Based Instruction Tutorial Session is designed for persons who have had little or no experience working with computer-based instruction (CBI). This session will give basic information that will help them get started in CBI. The topics for the session include:

(1) An examination of terms related to CBI and the differences between computer assisted instruction, computer-managed instruction, and computer test generation
(2) Characteristics of today's educational environment and the place of CBI in it
(3) Alternate CBI systems and characteristics of each
(4) Examples of successful CBI
(5) Questions one should consider before getting into CBI
(6) Where to get additional information or assistance concerning CBI

HOW TO CHOOSE A MICROCOMPUTER FOR EDUCATIONAL USE

Kevin Hausmann
Minnesota Educational Computing Consortium
2520 Broadway Drive
St. Paul, Minnesota 55113

ABSTRACT

Many people are realizing the tremendous potential that microcomputers have for education; however, it is becomming increasingly difficult to stay abreast of all the varieties of microcomputers currently available. One solution to this problem is to define the components that make up a minimal educational microcomputer system and then consider only those systems which meet minimal criteria.

The minimal system, as defined by Minnesota educators, consists of the following:

- An input device must be a typewriter keyboard and output a multiline monitor or printer.
- a permanent file storage must be some form of disk storage.
- The BASIC language must be supported.
- At least 12K of user memory must be availalbe, excluding operating system and language processor(s).

Since BASIC is the most often used language, an evaluation of BASIC language features and capabilities should be made. BASIC features typically considered include sequential file handling, random access file handling, chaining, special functions, matrix operations, formatted output, and good graphics commands.

Once the range of systems has been narrowed, ways to provide or acquire support should be considered. The following points were deemed important in the Minnesota plan for microcomputer support:

- One specific microcomputer (chosen through a bid process) should be available to educational agencies by a state contract.
- Instructional service support for the selected microcomputer should be defined and increased to the level currently available for timesharing.

- Continuous analysis and evaluation of hardware and software must keep up with the changing technology.

The technical evaluations and the invitation for bid used by the Minnesota Educational Computing Consortium are available from MECC Publications, 2520 Broadway Drive, St. Paul, Minnesota 55113 Ask for the 79-80 Microcomputer Report.

TURNING STUDENTS ON TO THE COMPUTER: THE INTRODUCTORY COURSE

Gary B. Shelly
Anaheim Publishing Company
1120 East Ash
Fullerton, CA 92631
(714) 879-7922

ABSTRACT

For as long as computing has been taught in high schools, colleges, and universities, there has been controversy concerning the first course in the curriculum. Some have advocated a high-level course dealing with algorithms and programming, while others have taken a computers-in-society approach. With the increased enrollment in this course, not only in colleges and universities but in high schools and even junior high schools, guidance must be given to teach a worthwhile course which can satisfy the large number of students from all disciplines who will be taking it.

This tutorial will present a suggested course content and methods for implementing this course at all levels of education. It will include justification for the subject matter which the author views as critical for the course and unique and effective ways in which this subject matter can be taught effectively.

INSTRUCTIONAL DESIGN AND COMPUTER EDUCATION

Gordon Stokes
Computer Science Department
Brigham Young University
Provo, Utah 84602
(801) 374-1211 Ext. 3027

ABSTRACT

The place of performance objectives, course organization, instructional strategies, and evaluation procedures will be discussed in this session. The instructional design of an introductory FORTRAN class will be the case history that illustrates the instructional design process in a classroom.

The presentation will give the participants enough information and examples to help them organize their own classes. The evaluation procedures will cover both formative and summative exams.

Some recent research results on individualized instruction in an introductory class will be presented, and their implications for instructional design will be discussed.

Invited Session

MICROCOMPUTERS IN EDUCATION

Chaired by Murali R. Varanasi
Dept. of Electrical Engineering
Old Dominion University
Norfolk, VA 23508
(804) 440-3741

ABSTRACT

Since the introduction of the first microprocessor in 1971, advances in the microprocessor field have so accelerated that universities have been faced with a serious educational challenge. To take advantage of the cost and flexibility offered by the programmable LSI devices, the digital designers have to be educated in software engineering; computer scientists have to learn digital systems at the gate and subsystem level. Even though microprocessors, they must be dealt with as any other computer--i.e., interfaces have to be built, buses have to be designed, and programs written. Therefore the following challenges arise:

1. How are design skills to be taught?
2. What sort of courses are needed?
3. What sort of laboratories are needed?
4. How does one introduce a Computer Science/Engineering student to application areas?

This panel will address the above questions from the perspective of experienced education, industrial, and government experts knowledgeable in computer education. Future applications of microprocessors will also be discussed.

Tutorial

PROGRAM DEVELOPMENT TECHNIQUES

A. J. Turner
Clemson University

ABSTRACT

Computer programs are often developed by users without considering the effort that will be required for someone other than the author(s) of a program to understand it, modify it, or correct an error. However, several techniques are available to assist in the development of computer programs that are easier to read, understand, debug, and modify. Since the techniques also facilitate the initial implementation of most programs, they are valuable even if a program is intended to be used without modification.

Techniques for three aspects of program development are considered in this tutorial: design, programming, and implementation. Topdown design is discussed as the basic approach to program design. Module independence, module function, information hiding, and the HIPO technique are considered as modularization criteria and paradigms to aid in the development of the design. The two programming techniques discussed that facilitate the development of program code are the use of pseudo-code and stepwise refinement. Also included are techniques for improving the readability of program code, such as structured coding and program formatting and commenting conventions. The use of iterative enhancement and module stubs are discussed as implementation techniques that complement the design and programming techniques and facilitate the use of a topdown approach to program development.

Emphasis is placed on the use of these techniques by a small implementation team or a single individual. Examples in the BASIC and PASCAL programming languages are included.

17

Business/Economics

COMPUTER SCIENCE AND MIS COLLEGE STUDENTS:

IS THE STAMPEDE FOR THE MONEY?

Eleanor W. Jordan
Department of General Business
University of Texas at Austin
Austin, Texas 78712
512-471-3322

INTRODUCTION

College programs in computer science and management information systems (MIS) are generally experiencing increasing enrollments in spite of declining or stable university-wide enrollments. Since the demand for DP professionals is currently very high, enrollment increases in these two areas may be an indication of the practical orientation of the '70s college students who supposedly left the liberal arts and demonstrations of the '60s to pursue safer career paths in a declining job market. If this is true, will colleges be producing more computer graduates in the '80s who are money-oriented rather than computer-oriented?

Industry is unlikely to be dismayed at the prospect of more practical computer science graduates. Bruce Gilchrist (5) predicts that the shortage of DP professionals will continue through the '80s mainly because educational institutions are not producing graduates with an adequate education for business applications. He suggests that computer companies spend less time raiding each other's DP employees and more time interacting with educational institutions in order to improve the DP personnel situation.

Several educators have joined industry recently in complaining that computer science education is often irrelevant to industry needs (2). Resulting recommendations have often been to add particular courses or stress particular languages (2), but there has been an increasing interest in new programs designed for the broadly defined fields of software engineering (7) or information systems (4).

Before too many new programs are developed, it might be valuable to consider whether it is possible to relieve industry's frequently expressed frustration with computer science graduates by making curriculum changes.

Paul Anagnostopoulos (1), a participant in the 1979 Brown University conference on the difference between software theory and practice, decided that the much talked-about software crisis is due mainly to programming personality defects. His suggestion that more attention be paid to the psychological aspects of programming and systems design is similar to Daniel Couger and Robert Zawacki's (3) suggestion that DP professionals consider behavioral aspects of personnel responsibilities as well as the required technical skills. In their investigation of more than 600 DP professionals (analysts, programmer/analysts, and programmers), Couger and Zawacki found that the stereotype of programmers as loners had considerable basis in fact: DP professionals reported significantly less need for interaction with others than all six of the other professional groups studied.

Are people who choose DP careers markedly different from people in general? Couger and Zawacki found that practicing DP professionals have lower social needs than others and concluded that this trait had important implications for how DP work environments and job requirements should be designed. If computer science and MIS students differ from other students in job-related attitudes and achievement motivations, it could have implications for program design, course design, and student advising in the computer science and MIS areas.

STUDENT SURVEY PROJECT

In spring 1977 I was involved in the development of an undergraduate MIS program in the College of Business Administration at The University of Texas at Austin. A strong computer science program already existed at the graduate and the undergraduate level, but our business school committee thought that the orientation of the computer

TABLE 1. Mean Ratings of Importance of Possible Considerations In Career Choice for Student Sub-Groups

Career Considerations in Rank Order of Importance	Business (N=326)	MIS (N=21)	Computer Science (N=20)	Liberal Arts (N=40)	Natural Sciences (N=45)	Engineering (N=15)
1. Personal Interest*	3.4	3.3	3.0	3.6	3.5	3.4
2. Personal Abilities	3.1	3.2	3.2	3.3	3.4	3.1
3. Career Potential	3.1	3.3	3.3	2.8	2.9	3.5
4. Job Security	3.0	2.6	3.1	2.4	2.7	2.7
5. Salary**	2.7	2.4	2.9	2.2	2.3	2.5
6. Amount of Time Allowed Outside Work	2.6	2.4	2.4	2.5	2.3	2.9
7. Flexibility	2.5	2.5	2.5	2.5	2.4	2.4
8. Variety of Job Requirements	2.4	2.3	2.3	2.4	2.2	1.9
9. Opportunity for** Community Service	1.8	1.5	1.3	2.2	2.5	1.5
10. Prestige	2.0	1.5	1.9	1.9	1.6	1.7

Items are measured on a five-point scale with 0=indicating "not at all important" and 4 indicating "extremely important."

Asterisks indicate that an analysis of variance resulted in a statistically significant difference among sub-group means at the following levels: *--$p \leq .05$, **--$p \leq .01$

TABLE 2. Mean Scores for Student Sub-Groups On Work and Family Orientation Factors

Factor	Business (N=326)	MIS (N=21)	Computer Science (N=20)	Liberal Arts (N=40)	Natural Sciences (N=45)	Engineering (N=15)
Mastery	20.1	20.3	19.0	19.2	19.6	19.7
Work	19.7	20.7	18.4	20.4	20.5	19.9
Competitiveness**	13.7	12.1	11.2	12.1	12.3	12.9
Personal Unconcern	10.3	10.3	11.2	10.9	10.7	10.1

The overall group means for these factors were very close to the Helmreich and Spence normative data for 1455 college students.

**An analysis of variance resulted in a statistically significant difference among sub-group means at the .01 level.

science undergraduate program was more suitable for aspiring graduate students, systems programmers, or scientific programmers than business applications systems analysts. The MIS program was designed to provide the education for a knowledgeable business user who can effectively define his information needs as well as analysts who can design the requested software without the usual conflict between user and DP personnel.

Fall 1979 was the first semester for the proposed MIS program. Since it is a computer program in the business school I was interested in whether the students attracted by the program would have work-related attitudes and motivations similar to business students or CS students. I therefore included students in two of the required MIS classes in a Fall 1979 survey project that measured attitudes from a large sample of business students at the undergraduate and masters level. A smaller sample of liberal arts, natural sciences (including computer science), and engineering students was also included for a comparison with the MIS group. The resulting sample included 467 undergraduates; the large majority were business majors in accounting, management, or marketing.

RESULTS

Determinants of Career Choice

Most of the students participating in the survey reported that they had decided on a career goal, even though the sample included about as many freshman and sophomores as it did juniors and seniors. Approximately half of the decided group had specific career goals, while the other half indicated that they had decided on a general area but were not sure about what their specialization might be.

On the survey students were asked to rate the importance of ten elements in their considerations for choosing a career field on a scale of 0 to 4. The ten career choice items included on the survey are listed in Table 1 in the order of average importance to the entire sample.

Personal interest in a career field had the highest mean rating for all student sub-groups except the computer science students and engineering students who tended to rate the importance of career potential more highly than any of the other items listed.

The generally high ratings given to personal interests in career choice deliberations are not consistent with a popular view of '70s students as obsessed with financial security, but the rest of the items in the top half all fit this view. Matching a career with personal abilities ranks second for most groups and career potential, job security, and salary follow.

To determine what reliable inter-group differences might exist in deliberations about career choice, I performed a separate analysis of variance for each of the ten items. Statistically significant differences among means were found for three of the items: personal interest, salary, and opportunity for community service. For all three of these items the mean for computer science students was at one end of the range of sub-group means and the means for liberal arts students and natural science students were at the other extreme. The mean for MIS students was closer to the other sub-group means for all three of these career choice items. For these three considerations in making a decision about a career, the MIS students appear to be more like other students than the computer science students. For the other considerations included in the survey, the computer science students, like the MIS students, appeared to rate each item of similar importance or lack of importance to the other subgroups.

Achievement Motivation

The survey also included four measures of achievement motivation. These measures were taken from Robert Helmreich and Janet Spence's (6) Work and Family Orientation Questionnaire since considerable evidence exists for the statistical reliability and validity of this instrument. The four factors identified in Helmreich and Spence's analyses are designated as work, mastery, competitiveness, and personal unconcern. The first two would seem to be highly relevant to realistic motivations of aspiring DP professionals in the rapidly changing computer environment: work is intended to measure the desire to work hard and mastery measures desire for intellectual challenge. The items composing the competitiveness factor are related to a desire to succeed in competitive, interpersonal situations. A high score on the personal unconcern factor indicates a relative lack of concern about the possible negative interpersonal consequences of achievement. In validation studies these factors have been found to be appropriately related to measures of scientific achievement, college grades, and income.

Computer science and MIS students, according to my results, are similar to business, liberal arts, natural sciences, and engineering students in terms of work, mastery, and personal unconcern (Table 2). In a comparison of the six sub-group means the results of an analysis of variance test was not statistically significant for any of these three factors. However a statistically significant difference among means was found for competitiveness. The highest mean factor score was for the business students and the lowest was for the computer science students; the mean score on competitiveness for MIS students was similar to the middle scores of liberal arts and natural science students.

SUMMARY AND CONCLUSIONS

The sample of computer science and MIS students participating in this survey was quite small so any conclusions that can be drawn from the results must be tentative. Where comparisons could be made to other studies the data did appear to be representative of college students and consistent with the Couger and Zawacki (3) study of DP professionals' personal needs. The additional implications of my study may then be worth considering at least for purposes of discussion.

For the most part computer science and MIS students appear to make career choices in a manner similar to other students and have similar achievement motivations. However, where statistically significant differences do exist, the computer science students were found consistently to have an extreme position, while the MIS students consistently indicated choices and motivations similar to the other student sub-groups.

Implications for Industry

If the results of this study in an academic setting are combined with the Couger and Zawacki (3) study, the impression of the DP professional is that of someone who has low needs for social interaction and little desire for interpersonal competition relative to other professionals or aspiring business executives. Computer science students are just as interested in an intellectual challenge as other students, according to my study, and have a greater need for personal growth than other professionals, according to the Couger and Zawacki study. But challenge and growth are apparently not defined in terms of social interaction or interpersonal competition. If the business executive understands this then it seems highly plausible that it would be possible to capitalize on the task orientation of the traditional DP professional and benefit from the lack of possibly destructive interpersonal competition. Some of the closed-door complaints that are sometimes heard about DP shops may stem from an assumption that the source of the isolation policy is a know-it-all egocentricity, when in fact it is likely to be a difference in task orientation.

The results of this survey related to MIS students seem very hopeful for a future reduction of the frequency of conflicts between computer experts and business managers. For all analyses where differences in motivations or importance placed on career choice considerations were found to be statistically significant, the MIS students tended to take a middle position. Hiring a combination of MIS and computer science graduates may very well result in a DP shop that has better intra-company relationships and still delivers the technical expertise that may require an almost exclusive orientation toward the task at hand.

Another hopeful note is that the programmer's big ego described in some of the stories of Gerald Weinberg (8) in The Psychology of Computer Programming doesn't seem to be prevalent among either MIS or CS students. Weinberg's observations may be more based on egocentricity than egotism.

Implications for Educators

Gordon Davis (4) has discussed the conceptual differences in the functions of information analysts and system analysts as a basis for designing curriculum in MIS and computer science. The results of my study provide some additional support for the distinction between broadly defined programs in information systems and the traditional computer science program. The middle position occupied by the MIS students on a number of career orientation and achievement motivation measures may mean that they will generally be successful in the liaison and managerial roles they are often placed in. Also MIS programs apparently attract a different type student than the computer science programs. As the number of MIS programs increase the number of graduates seeking DP positions may increase at a faster pace than expected.

The high salaries and eager on-campus industry recruiters would seem to be the most obvious reasons for the continuing increase in MIS and computer science enrollments, but that's not what these students are reporting. Personal interests are more important than salaries for all these late '70s college students. With a continuing increase in college graduates and a variety of computer educational programs, perhaps the software race will catch up with the hardware advances faster than the latest predictions indicate.

REFERENCES

1. Anagnostopoulos, P. "Software Crisis: Method or Psychology?" Computerworld, October 29, 1979, pp. 25, 28.

2. Cook, J. R., Gallagher, M. C., Johnston, M. A. "An Analytical Study of Industry's Computer Education Needs." Interface, Vol. 1, No. 1, Winter 1979, pp. 52-57.

3. Couger, J. D. and Zawacki, R. A. "What Motivates DP Professionals?" Datamation, Vol. 24, No. 9, September 1978, pp. 116-123.

4. Davis, G. B. "Information Systems Curricula in the Business School." Interface, Vol. 1, No. 1, Winter 1979, pp. 2-4.

5. Gilchrist, B. "Wanted: DP Professionals for the '80's." Computerworld, January 7, 1980, pp. 6-7.

6. Helmreich, R. L. and Spence, J. T. "The Work and Family Orientation Questionnaire: An Objective Instrument to Assess Components of Achievement Motivation and Attitudes Toward Family and Career." JSAS Catalog of Selected Documents in Psychology, Vol. 8, 1978, pp. 35-55.

7. Jensen, R. W. and Tonies, C. C. "Software Engineering Education--A Constructive Criticism." Proceedings of the Sixth Texas Conference on Computing Systems, November 1977, pp. 5B-7 - 5B-13.

8. Weinberg, G. The Psychology of Computer Programming. New York: D. Van Nostrand Co., 1971.

FORCE-FEEDING SPSS IN MARKET RESEARCH AND ANALYSIS AT HAMPTON INSTITUTE

Howard F. Wehrle, III
School of Business
Hampton Institute
Hampton, Virginia 23668
(804) 727-5362

INTRODUCTION

Hampton Institute (HI) is a small, prestigious, four-year historically black college that grants graduate degrees in education, nursing, and (in conjunction with George Washington University) engineering. As of September 1979, the School of Business included 15 faculty members and 792 students (the largest enrollment of any element of the Institute) who represented 36 states, the District of Columbia, and the Virgin Islands. The School of Business is actively pursuing preliminary actions toward accreditation of the undergraduate program by the American Assembly of Collegiate Schools of Business (AACSB), with the mid-range goal of an accredited Master of Business Administration (MBA) program.

COMPUTER EDUCATION FOR BUSINESS AT HAMPTON INSTITUTE

The focal point for computer education at HI is in the Division of Computer Science, Department of Mathematics, which offers a major in computer science. COBOL is the principle vehicle for teaching, although two APL terminals are available.

In the School of Business, a single computer course, "Computer Concepts in Business," has used FORTRAN. Until the 1978-79 school year the course was taught by a full-time visiting professor, an IBM employee. Upon his departure, the author volunteered to teach both sections of the course, continuing the somewhat restrictive program-writing orientation using FORTRAN, which was established by his predecessor.

Beginning in September 1979, however, the main thrust of the course was substantially redirected from a rather narrow technical orientation to a broader managerial orientation more appropriate to potential junior managers who are undergraduate students in business. This redirection included a new text, O'Brien's Computers in Business Management (7), which emphasizes business applications and problems with substantially less attention to detailed programming techniques. In addition, preliminary guidance from the Chairman of the Department of Management and Marketing had emphasized that FORTRAN is not necessarily the preferred language. Accordingly, minor preliminary study has addressed the possible introduction of the C language (2) since software compatible with the Harris PDP-11 (now being installed at HI) is available. This guidance recognizes the fact that the goal of the School of Business is not to train computer programmers or potential data processing managers, but to train competent junior managers with a broad overview of business.

Hardware available to the School of Business as of September 1979 included five keypunches, one terminal interactive with the IBM 370/168 computer at The College of William and Mary in Virginia at Williamsburg, and an IBM System 3 at Hampton Institute. The System 3 allows batch processing of Statistical Package for the Social Sciences (SPSS) programs.

THE CHALLENGE OF BUSINESS 428-01

On August 30, 1979, the author, primarily management rather than marketing oriented, was offered the opportunity to teach, as an overload, Business 428-01, "Market Research and Analysis." This opportunity was eagerly grasped, after only brief consideration. A conventional syllabus complemented the text and supported the concept of team determination, team leader selection, and research topic recommendation by the students. For the class of 59 students (34 juniors and 25 seniors) specific guidance was given orally at the same meeting: "This course is team-oriented. Accordingly, by the next class meeting please arrange yourselves in teams of not less than five nor more than seven members, at least one of whom must have completed 'Computer Concepts in Business' (to ensure a basic familiarity with the mechanics of computer programming, keypunch operation, and program debugging). Select your team leader, and provide the instructor a written list of the team members, designating the team leader, the individual with basic familiarity with computer procedures, and a list of three topics you propose for research. One of these topics, if deemed suitable, will be approved by instructor. If none of the first three is considered suitable, the team will be required to submit a second slate of three topics." (This requirement was never invoked, although the instructor was asked by members of several teams, "Why did you approve that topic? It wasn't the one we really wanted; we just added it on to meet your requirement for three topics." The response: "It was the only one on your slate which had any 'meat', the others were lightweights which would not have presented any challenge to you.")

By the second class meeting, eight teams were organized and moving out smartly; the remaining seven students wasted their first four class meetings fighting the problem but none dropped the course.

A sample milestone chart and work plan were distributed the first day of class; teams were advised the next requirements after approval of their research topic were successive submission of an hypothesis, work plan, milestone chart, and questionnaire(s) to elicit data appropriate to determine whether the hypothesis could be supported or rejected.

RESPONSE: CONDUCT OF THE COURSE

One may wonder, at this point, what need is there for computer support or what computer application exists? After topic approval specified questionnaire(s) would be administered to a random sample of not less than one hundred nor more than five hundred respondents.

Each team was issued (and acknowledgement made by signature of a student team member) a folder containing selected extracts from the SPSS manual describing the options and statistics available for SPSS procedures CONDESCRIPTIVE, CROSSTABS, FREQUENCIES, PEARSON CORR, and SCATTERGRAM, and a data binder containing an unexploded printout of a program operating on a sample of five with such data as last four numbers of social security account number, age, height, weight, shoe size, color of hair, and color of eyes. These data items, admittedly remote form most business application, were selected primarily both to demonstrate sample SPSS capabilities and to challenge the students to define analogous application to their project.

For further reference, two copies of the SPSS manual (5) were available: one chained to the counter in the computer center where card decks are turned in for batch processing; the other, the instructor's personal copy. (A third copy, recommended for purchase by the Institute library as a reserve reference, was not yet available at this writing.) The instructor (along with his SPSS manual) was also available to team members as a resource person/advisor.

The projects moved forward. Most classes were split between rather cursory lecture coverage of the assigned text and substantial time for team work and consulation with the instructor as required. Many questions arose, most of which were answered in the Socratic manner or by specific reference to assigned coverage in the text. "Be of good cheer and read ahead." Needless to say, these techniques roused substantial complaint from some of the class.

Sampling of the approved student topics is included in the appendix to this paper. One common complaint concerning these topics took the general form, "What does this have to do with market research and analysis?" The reply was standard: "The thrust of this course is precisely that of market research and analysis; while we are not attempting to ace the Nielsen ratings, or ascertain the marketability of Old Tennis Shoe Bourbon, you are exercising the techniques that would be applied in either of those two or a multitude of other examples."

STUDENT PROBLEMS

One substantial inhibiting circumstance in an otherwise orderly progression of learning by doing was the delayed distribution of the sample SPSS program by the instructor to each team. (He had his own problems in debugging what should have been a relatively simple, straight-forward program, owing to his lack of familiarity with job control language (JCL) and the evolutionary progression of SPSS from Version 6.0, which he had last used six years ago, to the current Version H.8.0)

As the semester moved forward, and the milestone for submission of completed reports approached, the instructor received successive feelers and ultimately almost frantic pleas to delay submission of the report. Although time was not available to do other than adhere to the schedule, as a small encouragement the teams were advised that chioce of date and order of presentation would be available to the teams in order of their report submission: the first team would choose first, second, third, position on any of the three days; the last team would have no choice.

Several teams complained, with apparent justification, about a severe imbalance of work; some team members were obviously not pulling their share of the workload. Since it had been emphasized that the same report grade would accrue to each team member, concern was expressed about the apparent injustice of the workers subsidizing the drones. So, peer ratings were required of all members of each team. These were compiled, reviewed, and, after consultation with a colleague more experienced than the author in personnel testing and mensuration, appropriately weighted before incorporation in the final course grade. (In addition to the report, four graded exercises highlighting application of text material were administered during the term.)

Immediately after submitting each report, it was comprehensively and severely reviewed by the instructor, as if it were the first draft of a professional report. The following day, the report was returned to the team leader, and after his persual, was discussed at length with him and selected members of his team. It was emphasized to each team that their oral presentation would provide an opportunity to recoup some of the cuts suffered on the written report.

Several teams wanted to revise and resubmit the written report, but this commendable response was rejected because the lessons learned from the severely criticized original report might be dulled. The preparation of oral reports would be inhibited, and such a requirement would place an inordinate burden upon the students at a time when preparation for final examinations should make major demands on their available time.

FINAL ASSESSMENT AND LESSONS LEARNED

Although the author considers that Business 428-01, "Market Reasearch and Analysis", was successfully (albeit rather painfully, for some students) completed, he learned some lessons which should significantly improve his second and subsequent conduct of the course.

First, completion of "Computer Concepts in Business" should be a prerequisite to Business 428. This requirement would tend to even the student workload, since each team member would have some experience in keypunching program/data cards; and would be able to participate more fully in preparation of basic data for machine computation.

Second, although Lehmann (2) provides a comprehensive review of elementary statistics as an appendix to one chapter, the statistics course required in all undergraduate sequences in business at HI should be a prerequisite to Business 428. This requirement would eliminate the tendency of some students to call up unnecessary statistical routines and fail to call up some useful routines.

Third, SPSS is a valuable research tool which all undergraduate students in business should learn to use. This limited application in "Market Research and Analysis" at Hampton Institute is the first step in that direction: it is a required course for all undergraduate marketing majors.

APPENDIX: APPROVED RESEARCH TOPICS FOR BUSINESS 428

Team	Subject
1	What effect does the Hampton Institute population have on gross sales of stores in the Coliseum Mall?
2	Declaration of majors: Why we choose the majors we do
3	Food additives and preservatives
4	Salt II Treaty
5	Gas rationing program in the area
6	Trade school is a good alternative to college
7	Hampton Institute's intended growth
8	How future computerized purchase will affect consumers and industry
9	Social Security

REFERENCES

1. Alexander, Daniel E. and Messer, Andrew C. FORTRAN IV Pocket Handbook. New York: McGraw-Hill, 1972.

2. Kernighan, Brian W. and Ritchie, Dennis M. The C Programming Language. Englewood Cliffs, New Jersey: Prentice-Hall Inc, 1978.

3. Lehmann, Donald R. Market Research and Analysis. Homewood, IL: Richard D. Irwin, Inc, 1979.

4. May, Phillip T. Programming Business Applications in FORTRAN. Boston: Houghton-Mifflin Co., 1973.

5. Nie, Norman H., et.al, Statistical Package for the Social Sciences, 2nd ed. New York: McGraw-Hill Book Co., 1975.

6. ______________and Hull, C. Hadlai. SPSS Batch Release 7.0 Update Manual, March 1977. Williamsburg, VA: Computer Center, The College of William and Mary in Virginia, 1977.

7. O'Brien, James A. Computers in Business Management: An Introduction, revised ed. Homewood, IL: Richard D. Irwin, Inc, 1979.

8. SPSS Pocket Guide Release 8. Chicago: SPSS, Inc, 1979.

SHORT-RUN FORECASTING
OF THE U.S. ECONOMY

William R. Bowman
Economics Department
U.S. Naval Academy
Annapolis, Maryland
(301-267-3156)

OBJECTIVE

The work of professional economists most often used by government officials and private businessmen lies within the realm of macro-economic forecasting. This art, however, is rarely acquired by students of economics due to the expense of developing large-scale econometric models of the economy and the high level of statistical knowledge presumed to be necessary in macro-economic modeling.

In response to this void, a research seminar in economics at the U.S. Naval Academy has been designed to offer undergraduate students a chance to build inexpensive, simplified forecasting models. These models are based upon economic theories, or extensions of theories, learned in previous courses. They are derived with limited knowledge of few, but powerful, statistical techniques rarely encountered by undergraduates. As part of a class exercise, each student develops one sector of a macro-economic model. These sectors then become integrated into a simplified model of Gross National Product (GNP) and inflation (as measured by the GNP implicit price deflator).

BACKGROUND

The art of forecasting the economy combines intuitive judgments with computer-based statistical models of aggregate spending behavior. The goal of these models is quite heroic, to say the least; one must model the decisions of groups of millions of individuals in the market place. These individual decisions, when aggregated, determine the level of production, as well as the aggregate price level of all goods and services produced.

The uses of macro-economic forecasts have been widespread. They are used by government policymakers when evaluating alternative fiscal and monetary proposals, and by private individuals when evaluating expected economic growth rates and inflation within defined sub-sectors of the economy. The number of the more well known large-scale models is limited to a handful, however, due to the enormous cost of building and maintaining the computer-based models. Some models, like the Data Resources Inc. (DRI) model, have as many as 200 stochastic equations and over 350 endogenous variables.

As indicated above, these cost considerations also limit how undergraduate schools teach this important aspect of economics. If macro-economic forecasting is taught, it is often done by having the student merely manipulate previously developed models. That is, they may plug in a hypothetical value for a chosen policy variable (say the corporation income tax rate) and observe the model's predicted effects on the aggregate economy. While this approach teaches students the sensitivity of economic sectors to public policy, it tells them nothing about the assumptions behind the model of the structure of the equations used in the model.

METHODOLOGY

Regression Analysis

The methodology used for introducing the student to the art of forecasting is basically one of learning-by-doing. The student is first given an introduction to regression analysis.(1) The theory behind the statistical procedure is discussed only with regard to the assumptions one must make when using regression analysis. (Formulae are not proven since the student is considered a user.)

Computers make it easy -- and fun -- for the student to do regression analysis from the very beginning of the class. Simple time-series practice data files are created and stored. These files are accessed and analyzed with the regression package Time Series Processor (TSP) that was originally written at Harvard and later modified at Dartmouth to run on a time-sharing basis.

The control statements used to execute the program are quickly and easily learned. Students may print out their data base in easily readable format, compute correlation matrices, and run preliminary regressions with simple control statements. (See Table 1 of Appendix A.) More advanced techniques made necessary by the problem of autocorrelated error terms can also be taught to the student with the TSP package. Date transformations such as discrete lags, logarithms, and first differences are easily computed. Their effects on the predictive power and autocorrelation can be observed by comparing the model results using transformed data with those of the preliminary regressions.

More advanced transformations, including the Cochrane-Orcutt procedure(2) and the Almon Polynomial Distributed Lag (PDL)(3), are then presented and used. The latter transformations are usually considered to be beyond the realm of undergraduate students; however, the learning-by-doing approach (made available by the TSP package) makes their use both possible and highly instructive. (See Table 2 of Appendix A for the control statements used for some of these data transformations.)

Model Design

With the statistical knowledge acquired through doing regressions on the practice data files, the student builds a simple regression model [Ordinary Least Squares (OLS) or Two Stage Least Squares (TSLS)] for a selected part of the larger macro model. The choice of explanatory variables is based upon economic theories discussed earlier in the research paper. The quarterly data base used to analyze each student's model is the Department of Commerce's data bank used in its Bureau of Economic Analysis (BEA) quarterly model. This data base consists of nearly 750 variables and transformations of these variables that may be used as endogneous and exogenous variables. Thus, it provides (at the time of this writing) the student with an exhaustive source of quarterly time series information for the period of 1949:1 through 1979:3.

The student selects independent variables that are deemed appropriate for the theory he has developed previously and creates mass storage data files for easy access. Much effort is then taken to derive a simple model whose explanatory variables have the expected sign and are statistically significant. (See the "estimated coefficient" and "t-statistic" for the explanatory variables in Table 1 of Appendix B.)

In addition, the student must work to obtain the best over-all fit of the data and minimize the degree of autocorrelation. (See the "adjusted R squared" and the "Dublin-Watson statistic" of Table 1 in Appendix B.) This process usually involves strong trade-offs between the latter two statistics and provides the student with a real world sense of forecasting problems rarely encountered by economic majors.

After numerous structural changes of the model, the student selects that model which most closely achieves two objectives. First, the model should have a high degree of explanatory power over the historical data period; second, its forecast errors should be minimized. The latter is diagnosed by dividing the historical data into a sample period and a forecast test period. The model is then used to generate predicted values of the dependent variable that may be compared with the actual values. This process is especially important during established turning points of specific cycles for variables classified as "cyclical." (See the actual, predicted, and forecast values of a selected variable in Table 2 of Appendix B.)

The Forecast

Once the structure of the model is chosen, the student uses his best judgment of the values of the exogenous variables during the forecast period. This period is defined in the short-run, for example 1979:4 through 1980:4, to minimize forecast errors. These values are selected based upon expectations of professional economists and business leaders concerned with macro-economic forecasting as reported in recent issues of numerous journals and magazines.(4)

The student then plugs these expected values, or a range of values, into his model to produce "conditional ex ante forecasts." These forecasts may be altered if student expectations cast doubt upon the likelihood of the model's results, i.e., the "judgmental forecasts." It is in this last phase that the student comes to appreciate the limits of computer-based modeling and the importance of personal, informed judgment, common to all large-scale forecasting models.

Once each student's structural model and forecast have been completed, the forecast of GNP and inflation is done by using a reduced form model with the exogenous variables and predetermined endogenous variables used in each student's own structural model. This step permits each student to discuss his research project with other class members, while providing each with an awareness of the inter-relatedness of their work.

CONCLUSION

By the end of the semester each student has become fully immersed in a highly technical research topic. Each has related previously learned economic theories to empirical model building using regression analysis on time series data. With the aid of the TSP software and the college's computer hardware, each student has acquired a sense of accomplishment rarely experienced at the undergraduate level. The blend of economic theory and computer-based technical analysis often opens the eyes of undergraduate majors to a whole new world of excitement in learning.

FOOTNOTES

(1) Cochrane, D. and G. Orcutt, "Application of Least-Squares Regressions to Relationships Containing Auto-Correlated Error Terms," *Journal of American Statistical Association, Vol.44 (1949)*, pp. 32-61. (See Table 3 of Appendix B for an example of the Cochrane-Orcutt adjustment factor.)

(2) Almon, S., "The Distributed Lag Between Capital Appropriations and Expenditures," *Econometrica*, Vol. 30 (1965), pp. 178-96. (See Table 4 of Appendix B for an example of the Almon Distributed Lag coefficients.)

(3) The text used in the course is: Chisholm, R., and G. Whitaker, *Forecasting Methods*, Homewood: R. D. Irwin, 1971.

(4) Three sources of information are most often used: (1) Federal Reserve Bank publications (e.g. St. Louis's *Review*, New York's *Quarterly Review*, and Boston's *New England Review*) and large commercial bank publications (e.g. Citibank's *Monthly Economic Letter*, Chase Manhattan Bank's *Business in Brief* and *International Finance*, and Morgan Guaranty's *Morgan Guaranty Survey*); (2) national business magazines (e.g. *Business Week*, *Forbes*, and *Fortune*); and (3) newspapers (e.g. *New York Times* and the *Wall Street Journal*).

Tools and Techniques for Instruction

HYPERTEXT - A GENERAL PURPOSE EDUCATIONAL COMPUTER TOOL

Darrell L. Ward
North Texas State University

Steve Bush
The University of Texas Health Science Center at Dallas

The Hypertext computer system is described in this paper. Although its applications are varied, the major emphasis here is the use of this system in the educational environment. The major features as visible to the student and instructor are developed. For the instructor, these features include the use of Hypertext as an organization and lecture presentation tool. For the student, Hypertext provides a model of information that mirrors a library yet permits a computer-assisted instruction approach as information is perused.

INTRODUCTION

Hypertext, as an information organizing facility, was first described in 1970 (1). The major emphasis of this paper will be the use of Hypertext in an educational setting. Hypertext has been implemented at The University of Texas Health Science Center at Dallas and is currently being used within the Medical Computer Science Department at that facility. The implementation details of this system will not be described, though it is appropriate to describe the computing environment within which Hypertext now functions.

Hypertext operates on a highly reliable, dual processor Tandem-16 minicomputer system. The Tandem-16 was designed to satisfy critical computing functions; thus the architecture reflects the design criteria with a high degree of redundancy. The total system philosophy is to run non-stop and, in fact, a single hardware failure does not crash the system or contaminate the data in any way (2). The implementation language is TAL (Tandem Application Language), a block-structured, ALGOL-like language. The Tandem-16 is capable of supporting page-mode terminals, which permit user interactions to occur on pages of data, much like pages of books.

With the above as background, the remainder of the paper will address the way Hypertext presents information to both the instructor and the student. Section 2 will describe the model of information that Hypertext presents to its users. Section 3 will introduce the instructor's use of Hypertext in creating lecture materials, presenting information in the classroom interacting with students, and maintaining current materials in the subject areas. Section 4 will demonstrate the student's use of Hypertext. This section will describe the environment with respect to one course, although general use by the student could benefit the total educational process. Finally, section 5 will summarize the ideas presented and describe areas of further research using the Hypertext tool.

MODEL OF THE HYPERTEXT ENVIRONMENT

The basic unit of information in Hypertext is a page; the user is provided an environment of pages and relationships among those pages. Each page in Hypertext is owned by an individual from the community of users and may be declared private, preventing other users from reviewing its contents. However, the philosophy of Hypertext is to provide an environment of information sharing; thus typically most pages are public, accessible to the community of users.

Each Hypertext user enters the system through a page designated as the top page or entry page. When the user is identified to

Hypertext and verified as a valid user, the top page of that user is immediately presented to him. The user is then in command mode and may:

1) edit information on the current page.
2) view another page via a menu selection format available on the current page.
3) view another page via an implicit relationship between the current page and the "next" page.
4) view another page by explicitly naming the page of concern.
5) establish a new page and formalize a relationship between the new page and the current page.
6) establish a relationship between the current page and some other already existing page.

The above operations are by no means exhaustive, but identify some of the major functions that relate to the educational use of Hypertext.

RELATIONSHIPS AMONG PAGES IN HYPERTEXT

Each page within the Hypertext system is given a unique page number that is available for display along with the informational contents of the page. This system-assigned number is an explicit method of establishing inter-page relationships. Any page can be created or altered to point to another page by inserting a "HYPERJUMP" to that page number. Any number of pages may be related to a particular page by linking them in the above described manner. The Hypertext system implements the linking by providing the user a numeric selection menu of pages (numbered sequentially from 1) which are accessible from the current page, permitting the user to merely select the page of interest by depressing the appropriate numeric key.

The above environment of pages is illustrated in the following example representing one user's top page and its set of possible relationships with other pages. (See figure 1 for a graphic model of the example.) In this example the top page provides for the selection of one of three other pages. The other pages include:

1) a page containing "things to do" with no additional relationships.
2) a page containing the current classes taught by the instructor (e.g. FORTRAN, COBOL, etc.) and additional access to the rosters of each of those classes.

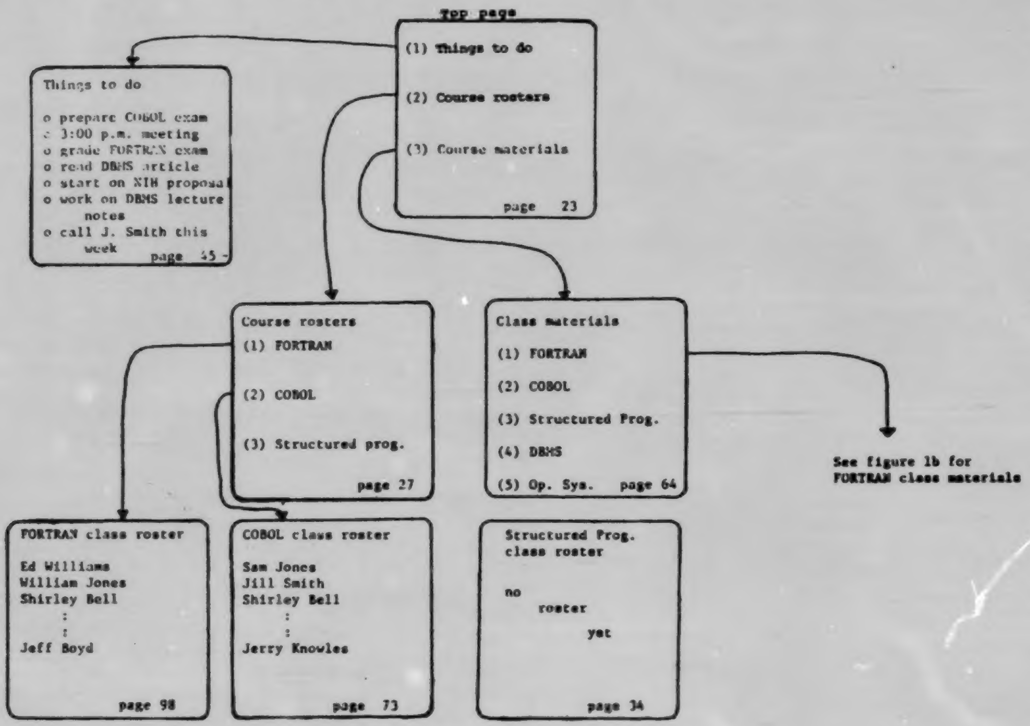

Top page
(1) Things to do
(2) Course rosters
(3) Course materials
page 23
Things to do
o prepare COBOL exam
o 3:00 p.m. meeting
o grade FORTRAN exam
o read DBMS article
o start on NIH proposal
o work on DBMS lecture notes
o call J. Smith this week
page 45
Course rosters
(1) FORTRAN
(2) COBOL
(3) Structured prog.
page 27
Class materials
(1) FORTRAN
(2) COBOL
(3) Structured Prog.
(4) DBMS
(5) Op. Sys.
page 64
See figure 1b for FORTRAN class materials
FORTRAN class roster
Ed Williams
William Jones
Shirley Bell
:
:
Jeff Boyd
page 98
COBOL class roster
Sam Jones
Jill Smith
Shirley Bell
:
:
Jerry Knowles
page 73
Structured Prog. class roster
no roster yet
page 34

Figure 1a. An example Hypertext organization for an instructor

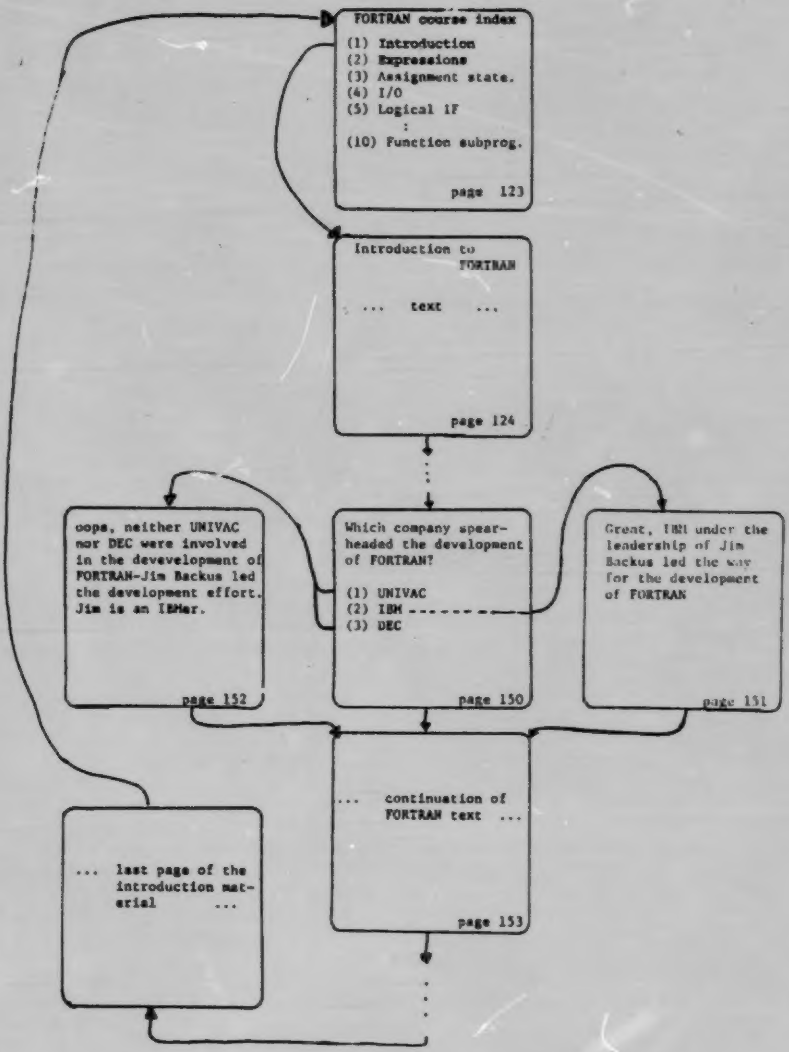

FORTRAN course index
(1) Introduction
(2) Expressions
(3) Assignment state.
(4) I/O
(5) Logical IF
:
(10) Function subprog.
page 123
Introduction to FORTRAN
... text ...
page 124
oops, neither UNIVAC nor DEC were involved in the devevelopment of FORTRAN-Jim Backus led the development effort. Jim is an IBMer.
page 152
Which company spear-headed the development of FORTRAN?
(1) UNIVAC
(2) IBM
(3) DEC
page 150
Great, IBM under the leadership of Jim Backus led the way for the development of FORTRAN
page 151
... continuation of FORTRAN text ...
page 153
... last page of the introduction material ...

Figure 1b. Hypertext example continued

3) a page containing courses already developed in Hypertext and the capability to access the index of each course via the menu selection process. Each course (only the FORTRAN course is illustrated) will consist of pages linked together with the possibility of many routes through the course materials depending on the user's responses.

TRAVERSING PAGES IN HYPERTEXT

Hypertext offers its users a variety of options for traversing its pages. This section will not attempt to describe all the options but will try to convey the atmosphere offered by Hypertext. There are several aids available to the users as they peruse Hypertext, including:

1) the ability to retrace pages already visited.

2) the ability to place landmarks (either in a temporary mode or permanent mode) on pages so that the user can directly return to a visited page of interest.

3) the ability to proceed forward through pages.

The traversal of pages can be illustrated via the previous example (the instructor's Hypertext). Consider a typical set of operations that an instructor might wish to accomplish:

1) add "grade COBOL exam" to the list of things to do.

2) drop "Ed Jones" from the roster of the FORTRAN class.

3) review the FORTRAN lecture material that will be presented in class the next day.

We will assume the user has successfully logged onto the Hypertext system and is currently positioned at the top page.

To accomplish 1:
-select the "things to do page" by depressing 1
-indicate the desire to edit the page (EDIT command)
-alter the page to reflect the added item (grade COBOL exam)
-back up 1 page (now positioned at the top page)

To accomplish 2:
-select the "Classes" page by depressing 2
-select the "FORTRAN class" page by depressing 1
-indicate the desire to edit the page
-alter the page by deleting the line containing Ed Jones
-back up 1 landmark (the top page is an implicit landmark, thus we are now back to the top page

To accomplish 3:
-select the "Courses" page by depressing 3
-select the "FORTRAN course" page by depressing 1
-review the index, select the appropriate topic of the lecture and depress that key
-review the pages of the lecture one at a time, altering the contents of any pages as desired and selecting the next page by depressing the RETURN key or function key 16 (the last page of the topic normally points to the index page)
-return to the entry page by going back 1 landmark

As the instructor peruses the lecture contents, there is ample opportunity to thoroughly test out all branching situations (via the back page function) and to alter any text that has changed or is incorrect. Also, the instructor can quite easily construct additional materials and link those at the appropriate point while traversing the course materials.

THE INSTRUCTOR ENVIRONMENT

The previous section hinted at some of the facilities available to individual instructors. This section will explore those aspects related directly to the teaching function and how Hypertext can assist that function in most instances. Previous work has shown the delete utility advantages of incorporating the computer into the preparation and delivery of course materials (3). The current Hypertext system significantly extends the previous work by providing the following additional functions:

1) the ability to traverse course materials in a very flexible manner(landmarks, backing up, skipping, etc.).

2) the facility for embedding, within a page, a call to an external program thus providing for an open ended system with respect to simulations, demonstrations, etc.

3) a dynamic, easy to use organization tool for creating and maintaining course materials.

4) a framework for combining materials to be presented in class with CAI materials to be taken by the student outside of the classroom.

The flexible approach toward visiting pages within the system allows the instructor to create course materials that suffice both for in-class presentation and for self-paced instruction. It permits the instructor to include additional material based on the response of the class. It also allows the instructor to leave more detailed explanations, examples, and problems for the students to discover on their own outside of class.

As as example of the above consider the pages of information shown in figure 2, a short segment of a FORTRAN course with branching possibilities. In an in-class presentation, the instructor can request, from the class, responses to the question contained in page 200. It may be apparent that the majority of the class understands the concept; thus the instructor can select to disregard follow-up of the question and go directly to the next block of materials to be presented, beginning at

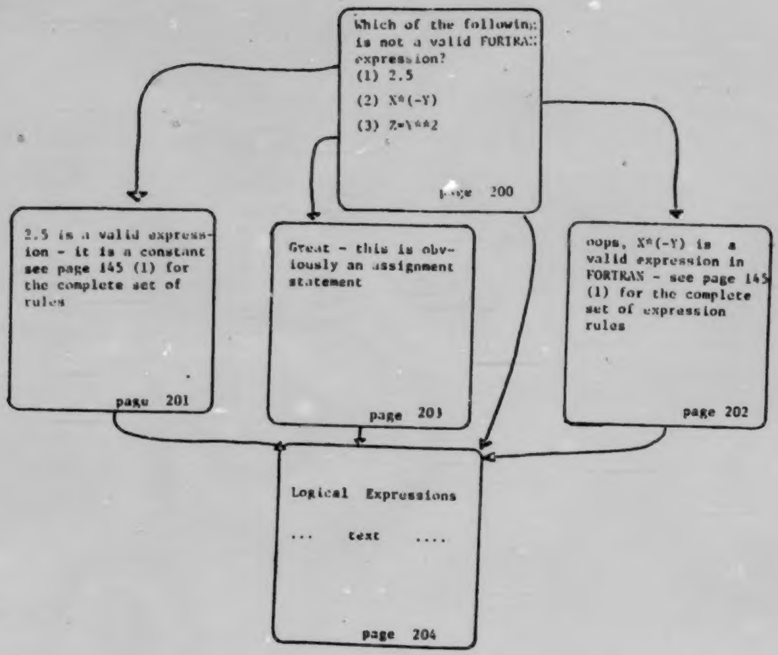

Which of the following is not a valid FORTRAN expression?
(1) 2.5
(2) X*(-Y)
(3) Z=Y**2
page 200
2.5 is a valid expression - it is a constant see page 145 (1) for the complete set of rules
page 201
Great - this is obviously an assignment statement
page 203
oops, X*(-Y) is a valid expression in FORTRAN - see page 145 (1) for the complete set of expression rules
page 202
Logical Expressions
... text ....
page 204

Figure 2. Typical text in an instructional segment

page 204. However, there may very well be some students that do not understand the question and interactions that transpired and do not stop the instructor to request additional information. This segment of the population could turn to the CAI mode of Hypertext and review the question on their own.

Finally, from the instructor perspective, a highly reliable computing system is essential. So far, the experience with Hypertext has been quite good; there have been no system crashes during the approximately 60 hours of in-class presentations given to date.

STUDENT ENVIRONMENT

Again, earlier work describing the classroom use of computers and student benefits apply in full to the Hypertext system (3). Briefly, these include:

1) no requirement for extensive note taking.
2) a clear, consistent prensentation medium.
3) an outside of class CAI facility for review of the in-class materials.

With Hypertext, as pointed out above, the functions available to the instructor are significantly extended. The student profits as well, as Hypertext allows him to access information the instructor has created in a friendly, reliable, and flexible manner. Although the student does not have the capability to alter this information, he may get a hard copy of any pages that are reviewed. Of course, the instructor may permit the student the ability to create pages. For example, consider figure 3 as a possible student view of Hypertext. The top page permits the student to organize each individual course of interest within Hypertext as well as his total educational environment. The student use of such a facility, with easy access to a reliable computing resource, would indeed promote a creative and exciting environment.

SUMMARY

Hypertext, as a tool to assist the presentation of information both in and out of class, has been presented. An overview of the Hypertext environment, as well as its uses from both the instructor and student vantage point, has been described.

Use of the computer in the classroom has been slow to develop due to the lack of supporting software and hardware. A reliable computing system with a creative environment such as

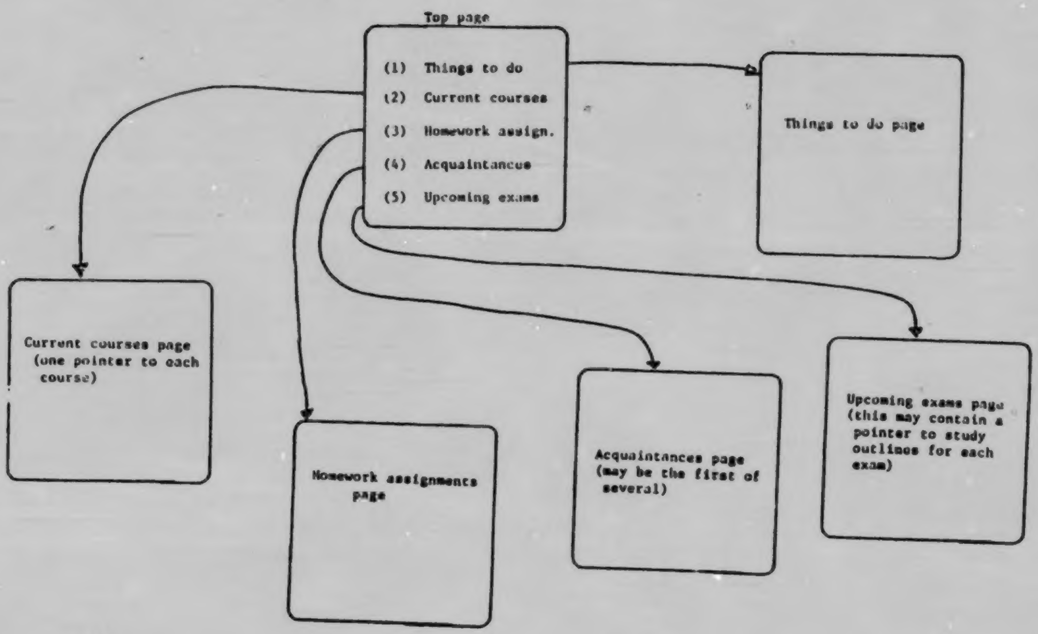

Top page
(1) Things to do
(2) Current courses
(3) Homework assign.
(4) Acquaintances
(5) Upcoming exams
Things to do page
Current courses page (one pointer to each course)
Homework assignments page
Acquaintances page (may be the first of several)
Upcoming exams page (this may contain a pointer to study outlines for each exam)

Figure 3, Example Hypertext for a student

Hypertext should facilitate the use of computers to assist the educational processes. Some areas, notably the reliability and interface, have been extended from earlier work (3); however, there remain some existing areas for future research. The ability to provide a user-oriented graphic facility embedded within a system such as Hypertext is an outstanding problem. Also, the ability to provide quality video display for a large audience requires additional work.

REFERENCES

1. T. H. Nelson. "No More Teacher's Dirty Looks." Computer Decisions, September 1970.

2. Tandem Corporation. Programming Manual. Cupertino, Ca. November 1977.

3. D. L. Ward. "A Computerized Lecture Preparation and Delivery System." Journal of Educational Technology Systems, Vol. 6(1), 1977-1978, pp.21-32.

4. R. Cheng. "On-Line Large Screen Display Systems for Computer Instruction." Proc. of ACM SIGCSE-SIGCUE Joint Symposium, February 1976, pp. 189-191.

5. W. Tracz. "The Use of ATOPSS for Presenting Elementary Operating System Concepts." SIGCSE Bulleting, 7:1, February 1975, pp. 168-171.

6. J. Rogers. "A Computerized Classroom for Instructor's Experimentation and Training." Computers in Education, O. Lecarme and R. Lewis (eds.), IFIP-North Holland Publishing Co., 1975.

A DYNAMIC PROCESS IN TEACHING TECHNIQUES

by

Jamil E. Effarah, Ph.D.
INFORMATICS INC.
21050 Vanowen Street
Canoga Park, California 91304
(213) 887-9121

ABSTRACT

Programs to teach people how to use computer products (hardware and software) often do not meet the needs of the diverse audience of users. The needs of one group may vary considerably from the needs of another. A lag exists between the hydra-like growth of the computer hardware and software industry, and the technology for teaching customers. This paper describes a practical teaching technique which has proved successful at Informatics Inc.

Teaching designs are based on identified procedures. This approach is represented by a network of relationships and interactions. The process starts with recognizing the audience segments for whom the courses should be developed, specifying their learning objectives, and planning a design customized to user needs. Brainstorming for ideas and selecting the most relevant comes next. After the form of presentation is drafted, at least two dry runs are conducted before representative audiences. Feedback is collected, analyzed, and evaluated; results are integrated into the evolving course. A field test at a customer site is the next step; more feedback is collected, analyzed, and evaluated. Finally, the validated responses are integrated into the courseware to produce a tested and useful educational program.

INTRODUCTION

Helping customers learn how to use software products is an important element in the design of those products. Some customers need technical training to install, implement, and support a product at their facility; others need to learn to use the product for reporting or processing data. Meeting those needs is the job of education development specialists in the Product Communications Group of Informatics Inc.

The Product Communications Group is a part of Technical Product Support; it includes technical writers, editors, graphics designers, and education developers. These communications specialists work together to document software products and to design courses for training customers. Their work begins early in the design of the product; software documentation and their teaching techniques are considered integral parts of the product package.

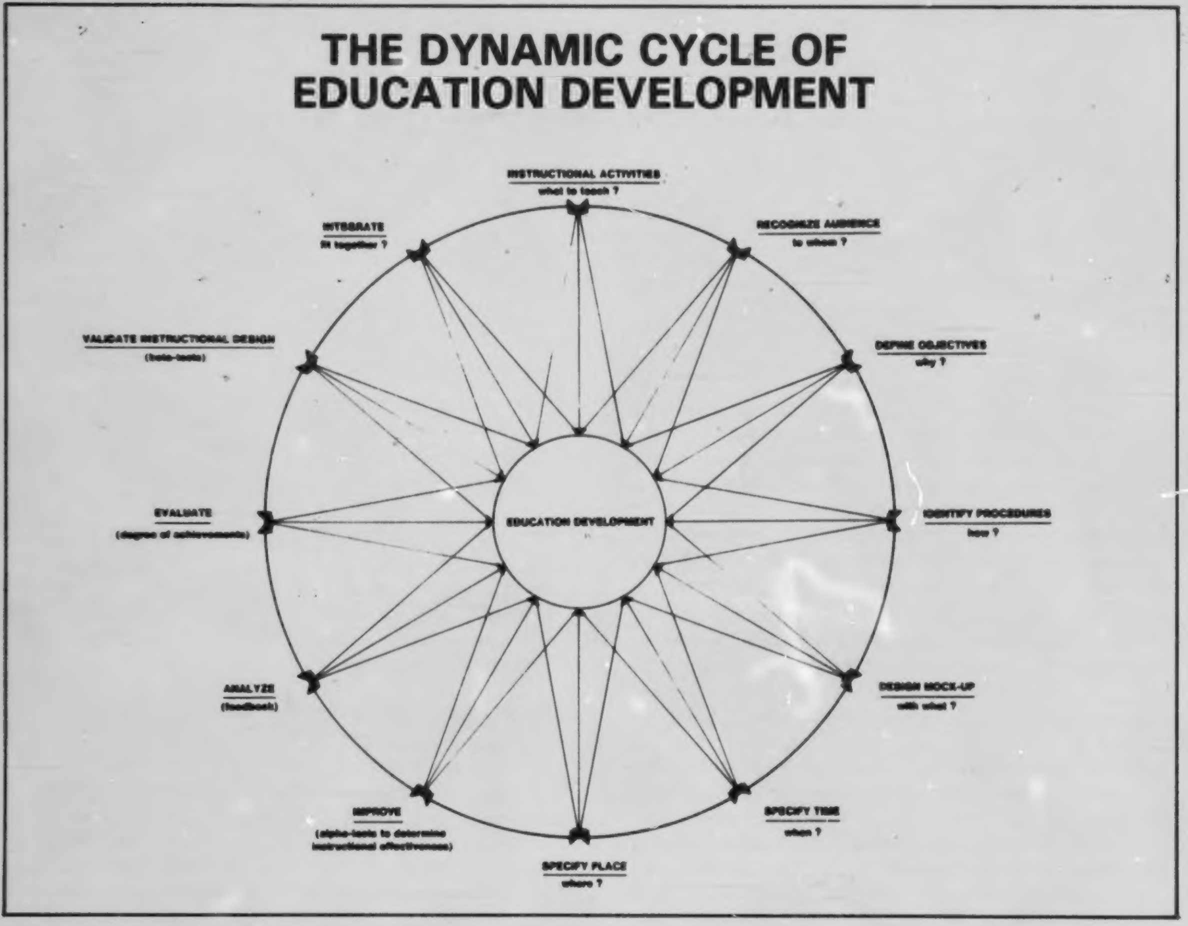
During the creation of learning activities at Informatics, the education development specialist is involved in a set of relationships and interactions that can best be illustrated as a network. This network design includes a systematic process for seeking relevant information to improve both course designs and teaching techniques. Figure 1 represents this process of continuous interactions and decision making.

IDENTIFYING THE USERS

In the computer industry, hardware or software products are manufactured to meet the needs of a wide spectrum of users. The first step in teaching techniques is to identify the audience segments of this spectrum — the people for whom the courses should be developed. For most practical purposes, two major audience segments, the end users and the technical support personnel, can be defined:

1. *The End User Audience.* This segment of users usually includes managers, clerical personnel, and casual programmers. The end users are, in general, not a data processing-oriented group because their primary interest is in their particular job; data processing is only a means to some other end.
2. *The Technical Support Audience.* This segment includes the support personnel, mainly data base administrators (DBAs), system designers, system programmers, and application programmers.

Depending on the level of complexity of the product, a plan should be developed for subdividing each audience segment into smaller segments with users of similar backgrounds and job requirements.



THE DYNAMIC CYCLE OF EDUCATION DEVELOPMENT

INSTRUCTIONAL ACTIVITIES
what to teach ?
RECOGNIZE AUDIENCE
to whom ?
DEFINE OBJECTIVES
why ?
IDENTIFY PROCEDURES
how ?
DESIGN MOCK-UP
with what ?
SPECIFY TIME
when ?
SPECIFY PLACE
where ?
IMPROVE
(alpha-tests to determine instructional effectiveness)
ANALYZE
(feedback)
EVALUATE
(degree of achievements)
VALIDATE INSTRUCTIONAL DESIGN
(beta-tests)
INTEGRATE
fit together ?
EDUCATION DEVELOPMENT

Figure 1

IDENTIFYING USERS' LEARNING OBJECTIVES

The second step is to identify the learning objectives which reflect the needs of each audience segment. For the two major groups of users, two sets of objectives are identified.

The end users need to know how the product can be used to their benefit and how to obtain the information which answers job-related questions and solves their problems. The following set of learning objectives might be developed to reflect end users' needs:

- To obtain job-related information.
- To acquire a basic level of skills essential to the use of the product.
- To explore and identify the capabilities of the product for solving problems.

The technical audience needs to know how to solve more complex problems — how to keep the product running efficiently, maximizing the benefits to their organization. The learning objectives developed for the technical support personnel might be represented by the following:

- To understand the difficult concepts and procedures which deal with the activities associated with the product.
- To explain and, where necessary, customize the product to meet the end users' needs.
- To identify the specific security requirements and the constraints that can be imposed upon the user for efficiency or security purposes.

PREPARING THE DESIGN PLAN

The educational courses are developed to satisfy the learning objectives which reflect the needs of the identified users. The procedures start with gathering information. The education development specialist interviews a diverse cross-section of specialized personnel; they contribute ideas and techniques relevant to course development, which might shape the product learning process. People's contributions are based on personal experiences in the field during interaction with similar users or on ideas from published sources, or they are gathered from colleagues.

The next procedure is analysis to find out how to fit together the collected ideas, how to look for the simplest and most relevant approaches to be used in presenting the capabilities of the product to the users, and how to produce a design covering all of the important and practical aspects of the product. Once the overall design plan is established, skills needed to accomplish each objective are specified and written down to form the basis for group discussion. In industry, they refer to this group as the document specification review committee. The committee members are representatives of the same groups who contributed ideas. They are called to review and comment on the design plan and the proposed specifications. As a result of the document specification review committee meeting, the education specialist revises the form and content of the evolving course and rebuilds it into a coherent design.

COURSE MEDIA

The media used in teaching are generally: handouts, overhead projector transparencies, the board, and the instructor's guide. This guide should be prepared carefully to provide consistent instruction. Wall charts are also important course media; they are developed for continuous reference during class session. Examples of wall charts used for IMS/VS environment data structure are shown in Figures 2 and 3. Figure 2 is intended for the technical group who normally looks at a hierarchical structure of logical statements from top to bottom. Figure 3 is intended for the end users, presenting the same concept in an approach that deals with building a relationship within a framework of reference, a readable left-to-right flow.

TIME AND PLACE CONSIDERATIONS

Time and place must be considered in course development techniques. The duration of the course, when in the sales/installation cycle it will be taught, and where it will be taught all affect the design of the course. The education specialist has to ask some questions about timing: Is the class needed before the installation of the product? If so, this may give the technical group an opportunity to become aware of what is expected during the installation procedures. Should it be taught during installation? When does the end user need the information? The education developer also has to consider location: Where will the course be taught? Will students have access to terminals? Will actual data bases be used for examples? The choice of customer site or a regional office has impact in these areas. These decisions are made by product management, marketing, and product communications after considering both technical and marketing needs.

IMPROVING AND EVALUATING THE COURSE

The process of course development starts with laying out a preliminary design resulting from selected ideas and committee discussion. To improve the course, dry runs are conducted before an audience representative of the group for whom the courseware is being developed. These dry runs help determine the effectiveness of the instruction, which is measured by the feedback collected after each dry run. Changes to the course are introduced, and the course is retested in another dry run before a new audience; more feedback is collected, evaluated, and compared to the results collected from earlier tests. This comparison measures the degree of improvement.

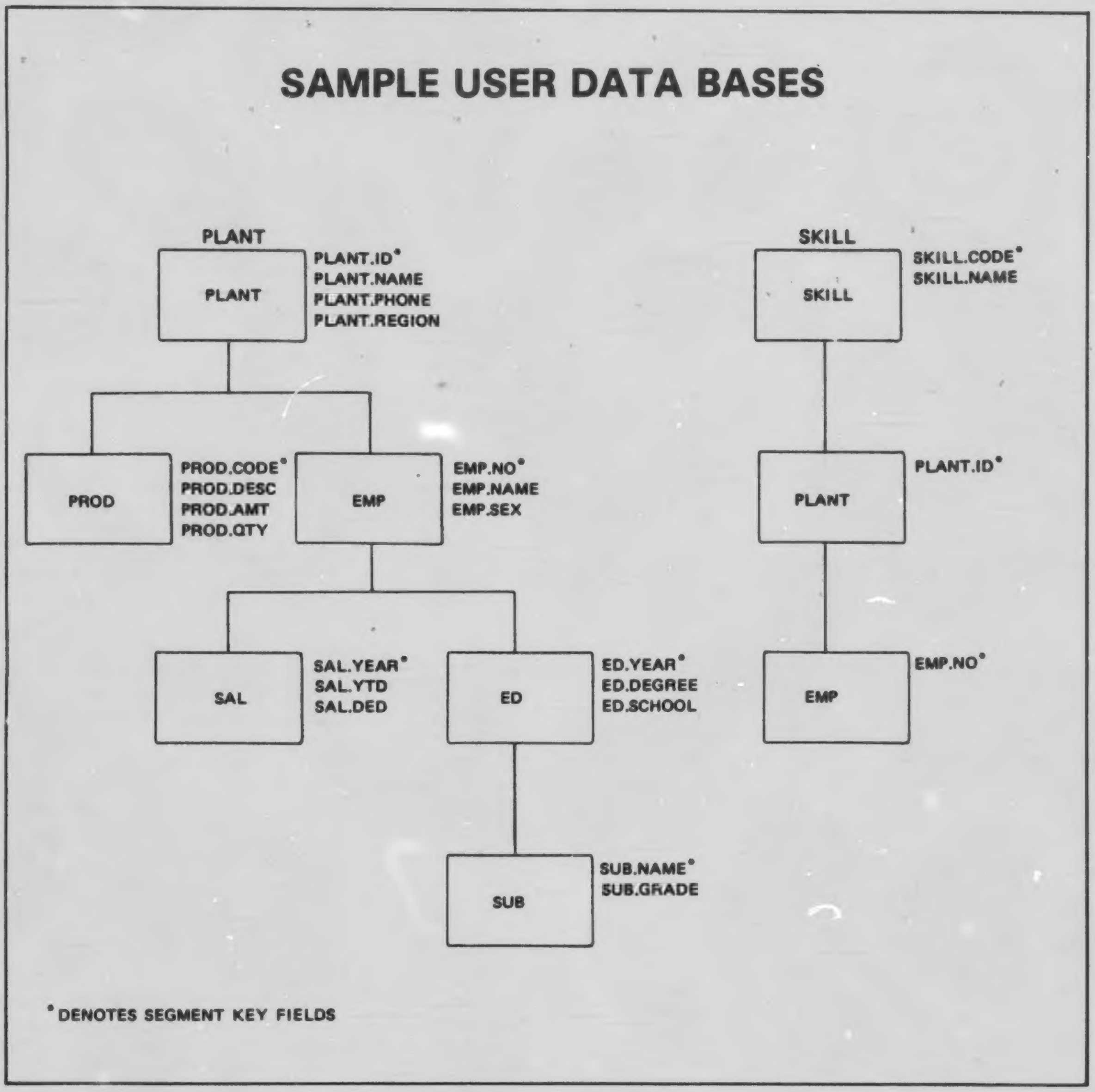

SAMPLE USER DATA BASES
PLANT
PLANT
PLANT.ID*
PLANT.NAME
PLANT.PHONE
PLANT.REGION
PROD
PROD.CODE*
PROD.DESC
PROD.AMT
PROD.QTY
EMP
EMP.NO*
EMP.NAME
EMP.SEX
SAL
SAL.YEAR*
SAL.YTD
SAL.DED
ED
ED.YEAR*
ED.DEGREE
ED.SCHOOL
SUB
SUB.NAME*
SUB.GRADE
SKILL
SKILL
SKILL.CODE*
SKILL.NAME
PLANT
PLANT.ID*
EMP
EMP.NO*
*DENOTES SEGMENT KEY FIELDS

Figure 2

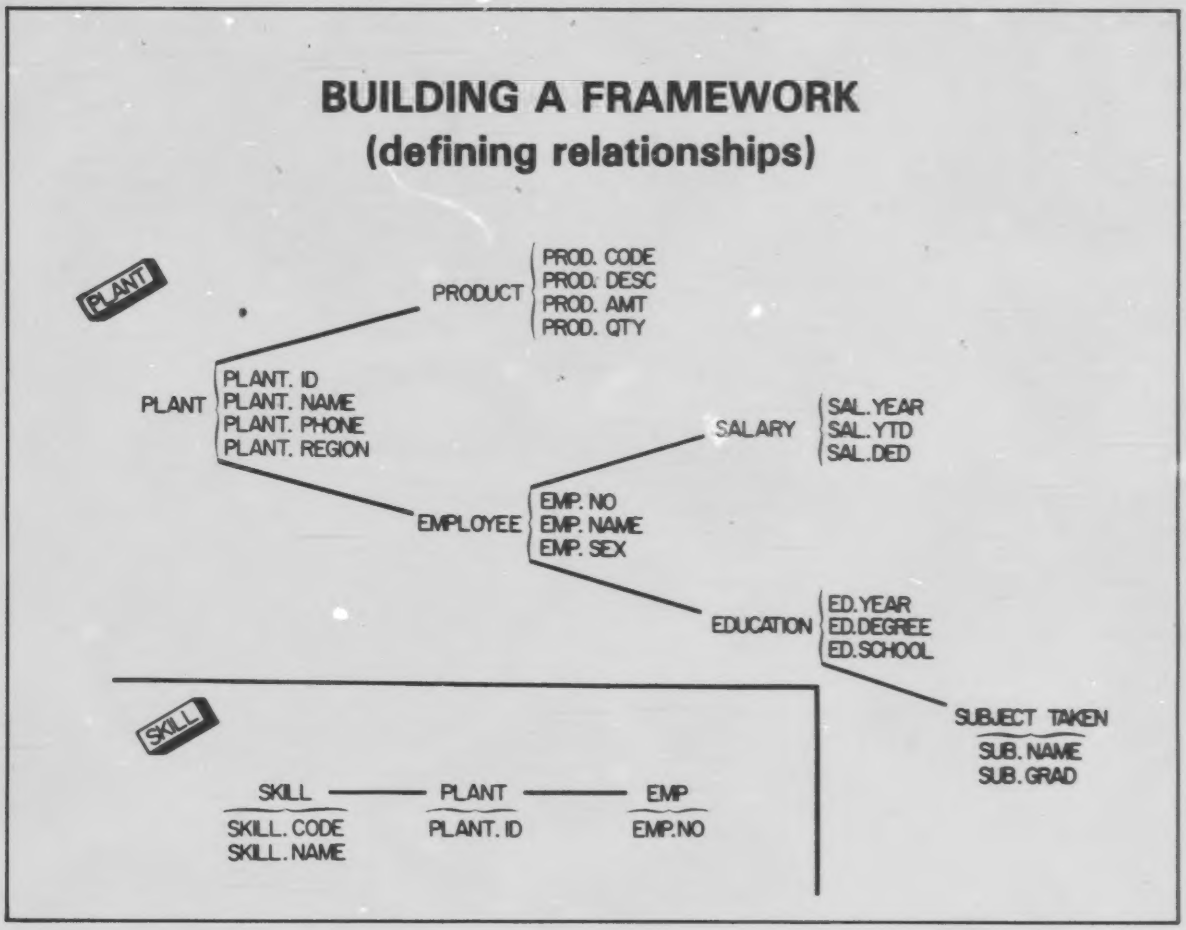

BUILDING A FRAMEWORK
(defining relationships)
PLANT
PRODUCT
PROD. CODE
PROD. DESC
PROD. AMT
PROD. QTY
PLANT
PLANT. ID
PLANT. NAME
PLANT. PHONE
PLANT. REGION
SALARY
SAL.YEAR
SAL.YTD
SAL.DED
EMPLOYEE
EMP. NO
EMP. NAME
EMP. SEX
EDUCATION
ED.YEAR
ED.DEGREE
ED.SCHOOL
SUBJECT TAKEN
SUB. NAME
SUB. GRAD
SKILL
SKILL
SKILL. CODE
SKILL. NAME
PLANT
PLANT. ID
EMP
EMP.NO

Figure 3

VALIDATING AND INTEGRATING THE INSTRUCTIONAL DESIGN

In industry, product communications group should not release the course to the field before a field test at a customer's location. After teaching the course to the intended audience at a customer site, the developer and course instructor collect written and oral comments that aid the developer in determining the degree of success in achieving the identified objectives for the course. The field test itself indicates that the minimum evaluation effort in producing a product learning activity is accomplished. Responses produced from the field test reflect the attitudes and feelings of the users to the course learning materials. In the process of validation, some of these responses are considered very useful to field instructors and are integrated in the instructor's guide to assure that field instructors are aware of users' expectations. Other responses may lead to the elimination of misconceptions in the course presentation. These changes are made before the final release of the learning document; however, no attempt is made to customize the instructional activities to every individual customer's needs. Examples are set to a typical and general application familiar to users. These examples can be modified, based on users' feedback, to become more realistic in the final touches for refining the courseware product.

SUMMARY

In summary, the instructional design technique that produces a successful learning activity should follow these criteria:

1. The learning activity or class material is designed to achieve specific learning objectives. Mainly, these objectives are to impart the knowledge of the product capabilities to the users and to enable them to use the product efficiently.
2. The learning activity is designed to offer replicable instruction, that is, instruction that can be taught in the field setting by any of the field instructors. The instructor's manual serves as a guideline to assure consistent instruction.
3. The learning activity is designed so its effectiveness can be tested and demonstrated. Typical problems, based on user needs, should be solved in class. Hands-on practical application of what has been learned should also be provided.

IN CONCLUSION

Software products and their instructional activities should be developed to work for the users' benefit. Users are encouraged to be a part of this dynamic process of education development. Their feedback is needed in order to learn how the product and its learning process can be made to work better.

Product education development techniques and the evaluation cycle have no time limit. They start with brainstorming but never end as long as the users are active in monitoring the product and its educational programs to service their job-related needs.

CONSIDERATIONS AND GUIDELINES FOR DEVELOPING BASIC SKILLS CURRICULUM FOR USE WITH MICRO-COMPUTER TECHNOLOGY

Robert M. Caldwell
Division of Educational Studies
Southern Methodist University
Dallas, Texas 75275
(214) 692-2347

The availability of low cost microcomputer technology is creating a revolution in education. Institutions of all types can now take advantage of the many benefits offered by computer-based education at a cost that is easily affordable for most. In addition, advanced microprocessor technologies interfaced with a wide range of audio-visual devices in just a few months have increased the capabilities of micros to include color, graphics, animation, and music and speech reproduction. In short, microcomputers have made available an extremely flexible and powerful teaching medium at a price that is finally cost-effective for most users.

In response to this growing interest in the use of microcomputers in instruction, several major publishers currently offer limited curricula in basic skills for delivery on microcomputers. A number of school districts such as the Dallas Independent School District and those districts affiliated with the Minnesota Educational Computing Consortium also have developed materials which cover a variety of skill areas. Because of the significant expenditures associated with developing these programs, however, most of them are limited in scope and employ a rather narrow range of teaching strategies and machine capabilities. In addition, few of the individuals currently engaged in instructional development have had much experience in using a highly interactive medium like the microcomputer. As a result, much of what passes for courseware today neither helps to develop higher level cognitive skills nor challenges learners to use higher order learning strategies. Most lessons utilize a drill and practice format in which the computer asks a question and requires the student to respond. Many so called tutorial programs are no improvement. They merely present segments of expository text and then display questions to test the student's comprehension of that text. This sort of of instruction has its use but implies to most educators that the computer's power lay only in its ability to present text and ask questions. This form of instruction serves to recreate the very worst of what presently occurs in a traditional classroom while ignoring all other teaching strategies that can help develop learning styles and learner independence(Garson, 1980).

To complicate this problem further, some companies are now developing authoring systems which will allow classroom teachers, students and curriculum developers to create their courseware through a process which utilizes templating and/or menu selection. To be sure, many educators will use these processes to create exciting programs which utilize the system's capabilities to its fullest. Many others, on the other hand, will duplicate the type of programs mentioned above; they will need help, guidance and education about what microcomputers can do and how instruction can be presented to take full advantage of the unique features of the new technology. They will need explicit guidelines that will help them devise ways to make contacts between the learner and the learning experience more meaningful, more efficient and more productive. The purpose of this paper, therefore, is to present specific guidelines for designing instructional programs that will be delivered on microcomputers so that those programs will use the capacity of the microcomputer system in a way that will develop in learners a range of cognitive skills and help learners develop useful learning strategies.

GENERAL FEATURES OF PROGRAM DESIGN

One of the most important factors inherent in programs delivered on computer-based systems is their ability to adapt instruction truly to the individual needs of each learner. With this in mind, then, programs of computer-based instruction should include the following general features (Caldwell and Rizza, 1979):

1. Learner Control over the instructional sequence is a feature of program design often ignored by instructional designers. Certainly there are situations in which a presentation of instruction must be linear. Certain subjects and concepts require it. In most other cases, however, consideration should be given to allowing the learner as much control over the learning sequence as possible. Options should be incorporated into the instructional sequence which allow for review of previous frames; for decisions about the type, difficulty level, and number of problems or exercises received; and for alternative branching routes that might lead to the accomplishment of lesson objectives in less time. In short, students should be given the opportunity to advance, review, and exit lessons except where such control defeats the purpose of the lesson. This ability of learners to pace themselves provides a degree of individualization not present in purely linear programs.

2. A system should be totally individualized and offer highly adaptive and responsive learning environments. By allowing self-pacing and individualized branching, learners are helped to select the pathway through the materials that is most appropriate for their needs.

3. Programs should be modularized and structured in coherent, hierarchial patterns. This type of organizational pattern allows for great flexibility in program implementation because curriculum materials specifically address each necessary skill in a defined content area in a manner that provides for development of skills not mastered or allows bypassing of skills which have already been mastered. This process can reduce student frustration and increase curriculum effectiveness.

4. All skills to be mastered should be carefully stated in performance objectives. The accuracy of a program is based on the specific definition of the objective in terms of performance competencies. Activities allow for precise diagnosis of skills already mastered, remediation in skill deficiencies, and exact evaluation of learner progress.

5. Progress should be measured in terms of mastery of performance objectives.

6. Strategies for diagnosis and prescription should be used. The efficiency of a program is due primarily to the diagnostic inventory made of the skills of each learner. This information can then be used to place them appropriately within the curriculum and to direct learners to the instructional material most appropriate to their needs.

7. Programs should be, when possible, multi-sensory in format.

SPECIFIC GUIDELINES FOR INSTRUCTIONAL DEVELOPMENT

Programs of computer-based education have been developed in a variety of designs and formats. Some use drills arranged in strands while others are built around a series of tutorial lessons. Many even incorporate all or most of the features mentioned above. Within these programs, however, are characteristics of instructional design that heavily affect the success of the instruction. The following is a description of some of the more common characteristics of lesson design that can contribute to instructional effectiveness and some that can seriously detract from it.

Creating Text and Graphic Displays

Many alternatives can be used to break the monotony of lines of text filling an entire screen:

1. Use graphics to box important sentences or paragraphs. Boxes made of lines or keyboard characters do nicely to alter the visual display of text on a monitor. Reverse highlighting and color can also be used effectively to accentuate visual displays. In addition, microcomputers can be connected to graphics tablets which make extremely interesting displays in almost any size or shape.

Whatever is used, it is crucial to interrupt continuous lines of text on a screen so that the screen does not look crowded and cause verbal overload in students. Too much text can have the effect of discouraging the learner, especially if he/she is a poor reader.

2. Allow the student control over the sequence of presenting text by breaking large portions of the text into discrete units or segments which the student can call up in sequence by merely pressing the space bar or some other key activated for this purpose. This procedure serves two

purposes: it allows learners to read at a rate that is appropriate for them, and it breaks the reading task into small segments so that the reader is not overwhelmed by page after page of text on the screen.

3. Animations, graphics, cartoon characters and other creative devices serve to create variety and interest in the display which appears on the monitor. Used creatively, these capabilities of microcomputer systems can contribute greatly to effective instructional programs.

4. Double space text material whenever possible to enhance the visual effect.

5. Use color to enhance the display or to highlight whenever possible. One program uses color-coded feedback to distinguish it from other text and to emphasize key concepts. Color is also useful in providing prompts and to direct attention to various portions of the screen.

Creating the Instructional Sequence

1. As a general rule, try to show learners rather than tell then. The overuse of exposition is the single biggest mistake instructional designers make. They feel as if they must write a lecture into each frame or provide complex directions, instructions or explanations when they can very simply use graphics or the ability of the computer to erase, rewrite, flash and even animate to make concepts clear. (The presentation which accompanies this paper illustrates this point with a number of examples.)

2. Make lessons as interactive as possible. Force learners to make choices; help them make decisions by providing them with options and alternatives. Simulation and dialog programs are the best instructional strategy for promoting interaction, but it can be provided through other means as well:

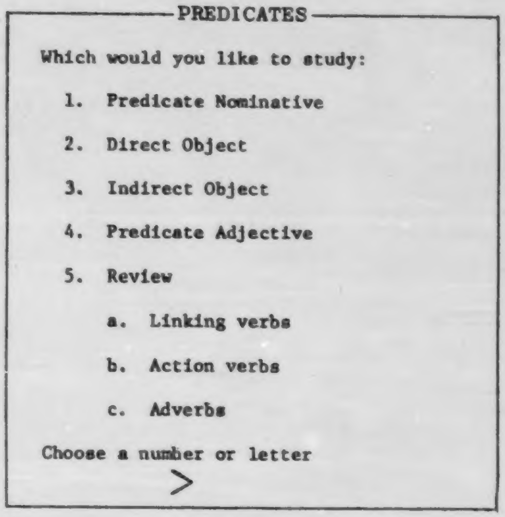
a. Menus

Learners may choose from a variety of options within a lesson or within a program by making choices from menus. These menus allow the learner flexibility to pursue his/her interests or to control the sequence in which topics are presented. For example, a language lesson which deals with predicates might allow students to access concepts which have no particular instructional sequence. Figure 1 illustrates just such a menu.

b. Performance Options

Learners should be given a variety of activities and choice of content. Tutorials should be accompanied by creative drills or instructional games that reinforce skills and information and enhance the interactive nature of the instruction. These games and drills can stimulate motivation by capitalizing on a novelty effect. Competition in the games can be against other students, the computer itself, time limits, or performance criteria set by other students (e.g. "the best score on this game to date is_____" or "the best time for this game is_____").

Figure 1



PREDICATES

Which would you like to study:

1. Predicate Nominative
2. Direct Object
3. Indirect Object
4. Predicate Adjective
5. Review
    a. Linking verbs
    b. Action verbs
    c. Adverbs

Choose a number or letter

>

A word of caution is appropriate here, however. In an attempt to bring novelty to their programs, some designers have defeated their own goals. In one such lesson students are encouraged to save a tiny man who is standing in a deep pit from a large heavy stone which is slowly rolling down a hill on the left side of the pit. This feat is accomplished by solving a series of math problems successfully. With each correct response the man moves up the right side of the pit toward freedom. If the learner fails to get the problem correct, however, the stone rolls closer and closer until it eventually falls into the pit and squashes the poor fellow. In an observation of this sequence with several children, a number of them chose to see the man get squashed with the full knowledge that to do so they must fail every single math problem presented to them. This exercise, therefore, did little to develop math skills but did much to distract the learners from the true intent of the lesson.

c. Prompts

The power of computer-based instruction resides in its ability to shape learner behavior toward learning outcomes in a way not possible with most other media. An important factor in shaping behavior is the use of prompts. Designers who ignore the use of prompts often turn the instruction into a guessing game. For example, a number of math programs currently available present problems then simply respond to the learner's incorrect response with a "No, try again." This type of feedback gives the learner no information about how he/she is doing; it only promotes guessing until the correct answer is discovered. Prompts such as, "Too high, try again" or "Remember, carry the ones," on the other hand, provide learners with guidance toward the correct answer. Without them, interaction becomes meaningless. More about prompts will be said later in this discussion.

d. Task Description

Learners are very often misled about what it is they are supposed to do or how they are supposed to respond in a given exercise because the instructional designer has confused the task for them. The following is a perfect example: In this lesson the learner is presented with the task of distinguishing complete and incomplete sentences:

The store closed early.

Type comp (for complete)or inc(for incomplete)

The task here is clearly discrimination between complete and incomplete sentences, but it is confused by asking the learner to type "comp" or "inc." The instructional sequence is marred not only by a poorly designed drill and practice but also by a poorly conceived response format. An improved version would simply ask the student to respond by typing a 1 for complete (or a c) or a 2 for incomplete (or an i), thus focusing attention on the discrimination task rather than the typing task.

Task confusion can also be lessened or eliminated by providing learners with a sample exercise before they are engaged in a drill or a quiz.

Input and Response

1. The method an instructional designer chooses to facilitate learner input and response is largely a function of the type of learning outcome desired. Response modes most commonly include typed response, touch response or light pen response. The first criterion for choosing one of these modes is whether the response will help reach the objective of the lesson. One illustration of this rule is in the example cited earlier in which students are asked to discriminate between complete and incomplete sentences. In this case a typed response was inappropriate to the objective of the lesson. In fact, typed responses in general seem prone to high error rates because of th[illegible]ature of typing tasks (Caldwell, 1974). [illegible]ped responses also become time consuming especially if learners are unfamiliar with the keyboard. This problem can have the effect of distracting learners from learning the concept being presented. A related problem is that many errors in typing are judged by the computer to be incorrect responses, and therefore students are routed to remedial sequences or to segments of the lessons they have just been through.

Similarly, if typed responses are not carefully cued, learners will often be forced into closed loops; that is, they try to find the correct answer by typing random responses. When this problem occurs learners have little idea about what is expected of them and become frustrated trying to find the response that will advance them to the next frame. This common error in instructional design has an extremely negative effect on learners.

Some advantages can be found in the typed response, however, which are not present in forced choice response modes (Caldwell, 1974):

a. Learners seem to express more favorable attitudes toward writing and the use of verbal language.

b. Spelling skills and the ability to generate language improve.

c. Personalized feedback is made possible through direct entry of student names.

2. As mentioned earlier, response by direct entry should not be required of a student without an example or a practice exercise. This can prevent learners from making errors in the practice exercises. These errors can become important, particularly if one is using response data to branch learners or to do item analysis.

3. A number of other helpful suggestions can be found in the guide to developing instructional software developed by the Minnesota Educational Computing Consortium (1980). Some examples from that guide follow:

a. Be consistent in the way questions are asked and the required format for responses. Do not code answers unless it is necessary. Request the responses of YES or NO rather than coding responses such as 1=YES and 0=NO. When processing these answers check the first character to determine if it is "Y" or "N". If it is neither erase the student's answer and re-ask the question.

b. Ask questions while the information needed to answer them is still on the screen. Don't ask a question then change the screen to ask another one leaving the choices for response on the screen just erased. If the student must choose from a fixed set of options, then those options should remain on the screen throughout the quiz sequence. In some cases it is helpful to allow students to go back to formerly displayed frames to refresh their memories. This review is accessed by means of the HELP key.

c. Put options for a question in a column. Do not imbed them in a sentence. Options having multiple words should be coded with numbers or letters.

d. Answer checking after accepting input deserves specific consideration. Don't ask a question and expect a very specific answer. For example, if the question is "What is the capital of Minnesota?" some learners might answer, "The capital is St. Paul." If the response judging is too rigid and is looking for just "St. Paul" this student's answer would be wrong. It is therefore very important to do keyword matches rather than a match on the whole answer entered.

Also, when looking for correct answers, consider how many times to allow a student to miss a question. A typical method used by many designers is to ask a question a maximum of three times. This way, the learner has the opportunity to be prompted twice. Any more than three tries seems to indicate that the student has not mastered the concept and it is best to move him/her on. However, it is a good idea to ask the same question again later.

4. The number of questions presented during a formative evaluation is the subject of considerable debate. Essentially, the number of questions presented in any one exercise is dependent upon the objective being assessed. As a general rule, however, five (5) is a number that seems workable for most learners. It is a sufficient number to assess progress toward mastery of the objective and requires just enough time so that the exercise itself does not become tiresome for the learner. Five items is a good number for generation of item pools also if one uses the guideline that three times the number of items should be written as are presented in the exercise.

5. One must also be wary of the type of information learners are permitted to input. For example, a common practice in many computer-assisted lessons is to ask, "What is your name?" or "What do your friends call you?" The purpose of such a question, of course, is to use this information later to provide personalized feedback or to use the learner's name in the content of the presentation (e.g. "MEMO to: Johnny"). This practice can sometimes have disastrous effects if the learner decides to get cute. One student, for example, was observed to enter "idiot" to just such a question as those cited above. All successive feedback after that resembled statements such as, "No, Idiot, try again" or "Good work, Idiot." While this type of response can delight learners, its overall effect is one of distracting them from the purpose of the lesson.

Reinforcement

Considerable space has been devoted in this paper to suggestions regarding responses to various types of imput. The section dealing with prompts serves as a good example. Generally, reinforcement should be specific and directed at providing information that will help shape the learner's behavior toward the desired learning outcome. As an illustration, consider the following sample sequence:

----Add - ing to the word----

Come>

(Here students are required to transform the word by retyping it with an -ing ending.) A correct response on the first try would warrant positive reinforcement such as "great, super, well done, excellent." These and other reinforcing statements can be generated randomly from a list of 20 or 30 possible statements. Incorrect responses on the other hand, should follow a pattern of prompting which should lead the student to the correct answer. For example:

First incorrect response: Come> comeing (drop the e). The student is cued by reminding him/her of the rule under examination (drom the e).

Second incorrect response: Come> comiing (come + ing). Here the learner tries a different response. In the second prompt the computer system animates the e in come and it drops from the equation then the -ing moves into place.

Third incorrect response: Come> Coming . If on the third try the student still does not type the correct response, the correct answer should be given, and the student should then move on to the next problem. This type of reinforcement pattern contributes to meaningfulness of the material to be learned because it:

a. Provides specific information that helps guide correct learner behavior toward achieving the desired outcome.

b. Reduces the frustration often experienced by learners in CAI programs that simply provide a "no" response to an incorrect answer. A simple "no" is unsatisfactory because it provides no specific feedback that helps the learner to discover correct responses. Instead learners are forced to guess until the correct answer is found (Caldwell, 1979).

Reinforcement strategies depend largely on the nature and type of learning being attempted. Reinforcement, however, can be made more meaningful if it is personalized and specific to student response sequences. (e.g. "You did well, Allen, but would you like to review prefixes before going on?")

In summary, the design of instructional programs should incorporate various teaching strategies to add variety to an overall curriculum and to address cognitive processes at all levels of that domain of learning. Also, programs should use to their best advantage the features of color, graphics, and animation offered by microcomputer technology. Exciting new developments in speech and sound peripherals also exist and should be incorporated where relevant and appropriate. Also, learner control over the instructional sequence allows for individual pacing and better opportunities to achieve mastery through branching, diagnosis and remediation. This flexibility can add significantly to computer based programs if they are designed and implemented carefully.

The scope of this presentation limits discussion of the many subtle variables which contribute to effective instructional design, but attempts to begin an organization of some of the features which seem to contribute to successful programs. It is hoped that it might stimulate other authors and designers to share their rationales and experience in designing high quality programs of computer-based education.

REFERENCE NOTES

Caldwell, R.M. "The Effects of Selected Strategies for Teaching Reading to Non-literate Adult Learners Using Computer-Based Education." Paper presented at the annual meeting of the American Educational Research Association, San Francisco, April, 1979.

_________. "Evaluation of a Program of Computer-Assisted Reading Instruction for Semi-literate Adolescents." Paper presented at the annual meeting of the American Educational Research Association, Chicago, March, 1974.

REFERENCES

Caldwell, R. M. and Peter J. Rizza. "A Computer-Based System of Reading Instruction for Adult Non-readers," AEDS Journal, Summer, 1979, 12,4.

Garson, P. W. "The Case Against Multiple Choice," The Computing Teacher, February-March, 1980, 7,4.

Minnesota Educational Computing Consortium User Services, A Guide to Developing Instructional Software for the Apple II Microcomputer, St. Paul, Minnesota: The Minnesota Educational Computing Consortium, February 15, 1980.

Invited Session

RESEARCH ON MICROCOMPUTER USES IN EDUCATION

Chaired by David Kniefel
New Jersey Educational Computing Network

THE AFFECTIVE AND COGNITIVE EFFECTS OF MICROCOMPUTER-BASED SCIENCE EDUCATION

Ronald E. Anderson, Daniel L. Klassen, Thomas P. Hansen, Minnesota Educational Computing Consortium

ABSTRACT

Microcomputers are used increasingly to deliver and enhance science instruction, but the impact of these new technologies has not been extensively studied. An experiment was designed to investigate the effects of a brief CAI activity on student attitudes, beliefs, and knowledge. Three hundred fifty high school students were tested and retested before and after taking a 20-30 minute lesson on water pollution. The instructional material was all delivered by an APPLE II microcomputer using high resolution graphics and some color variations. The findings support the claims that: (1) even a very brief CAI exposure can be instructionally effective; (2) understanding about computers, i.e., computer literacy, can be improved as a consequence of such CAI: (3) students become attitudinally more positive about computers from microcomputer CAI: (4) graphic enhancements may not improve student responses; (5) system malfunctions may revise student conceptions of themselves as well as computers.

The micro offers a vast potential with some possibly questionable effects. Since it allows new kinds of person-computer relationships, we can not presume to know its ultimate impact on individuals.

THE EFFECTS OF MICROCOMPUTING ON LARGE CENTRALIZED TIMESHARING

Kent T. Kehrberg, Minnesota Educational Computing Consortium

ABSTRACT

Minnesota schools have already acquired nearly 1,000 APPLE II and other microcomputers, even though they have had access to a large central computer via a statewide educational network. This situation offers opportunity to research some important questions such as: (1) How do people justify their acquisition of microcomputers? (2) How are micros used in the schools? (3) Does microcomputer acquisition substitute for or foster timesharing use of a centralized facility? The MECC research serves as the basis for projection of the role of microcomputers and instructional computing in the short-term future.

DISCUSSANTS

Harold Peters, Associate Director, CONDUIT
John Castellan, Jr., Indiana University

Computers in Humanistic Studies

CREATIVITY THROUGH THE MICROCOMPUTER

George M. Bass, Jr., Ph.D.
Assistant Professor
School of Education
College of William and Mary
Williamsburg, Virginia 23185
804-253-4289

> "Creativity is a battle
> against fixed attitudes."
> (Raudsepp, 1977, p.55)

In many ways teaching a college course on creativity is very similar to teaching students how to use a computer. Both sets of students come with many preconceived notions about the subject. Usually these attitudes have not been built upon direct experience with computers or creativity experiences but on hearsay and vague expectations. This lack of experience has often led students to be fearful of their own ability to be creative or to master the computer. Yet a willingness to break free from these frozen views and to see ideas and problems from a new perspective is needed in both areas for the student to succeed.

During the fall semester of 1979, I taught an advanced education course on creativity. This course was designed to be a seminar for students who had successfully completed an introductory educational psychology course. My overall goals were to present students the necessary background about creativity as it has been studied by psychologists and educators; to introduce them to instructional strategies which may help develop creativity; and finally, to increase their own creative abilities. Eleven students took the course.

COURSE ACTIVITIES

In order to distinguish this course from just another elective, I wanted to use activities not typically found in other courses they had taken. Entering national contests, participating in creative growth games, working on word puzzles and logic problems, reading and solving detective stories, producing poetry and written compositions, and using a microcomputer were introduced throughout the semester. (More traditional activities such as assigned readings, class presentations and projects, and a final position paper were also used to increase the learning process.)

Because the microcomputer is such a new educational technology, it is an ideal way to break the chains of student expectation and release any hidden potential. Yet the fear of the unknown or the exaggerated visions of computer power (such as HAL in 2001) were a real concern. In order to reduce this possibility, I began the first class with TLC -- Tender Loving Computer. Other exercises using the capabilities of our Apple II microcomputer were later interspersed throughout the course.

MICROCOMPUTER ACTIVITIES -- OBJECTIVES AND RESULTS

1. Course Syllabus

Since this was the first time the creativity course had ever been taught, I felt it necessary to give students my course objectives, assignments, and requirements during the first session. However, I wanted to communicate this basic information in a nontraditional way as mentioned earlier. I also wanted the students in the class to meet one another and to learn of any personal objectives each individual might have for the class. To accomplish these objectives, I wrote a program on the microcomputer to interact with each student. After requesting the student's name and some background information (interests, reasons for taking the course, etc.) and engaging him in chit chat, the program presented relevant information about the course and then introduced the student to the capabilities of the microcomputer. Using a variety of programs currently available for the Apple II, the students were presented with computer graphics, text manipulation, and game playing experiences, e.g., Rock/Scissors/Paper, Dragon's Maze, Star Trek.

The results of this experience were quite positive. The students had never seen a microcomputer before and were surprised with the variety of things it could do. As each student took a turn, the rest of the class crowded around the color monitor to watch the action. They got to know each other through the typed conversation with the Apple II. They also got to see many of the programs available to them outside of class. (For two students this introductory session led to a semester-long journey with Star Trek after almost every class!) This computerized syllabus generated interest and set the tone for an unusual class.

2. "How I Spent My Summer Vacation" Assignment

One of the most common assignments that teachers give when students return to school in the fall is to write a theme on how the student spent his summer vacation. In order to get the students in the class to begin thinking of novel ways to solve problems or accomplish tasks, I gave the class this typical task, but asked them to solve it in a creative way. When they next came to class with their creative solutions, I shared with them one solution using the microcomputer. I programmed a story which contained an overall structure but required the addition of certain words and information from the user to compose a complete tale of how the Apple II spent its summer. Besides showing how unique paragraph-length stories can be created using a microcomputer, this exercise laid the groundwork for a future class discussion on form versus content in creating something new.

While the students' creative solutions to this assignment were indeed varied (a picture scrapbook, a summer job advertisement, "first grader's" paper, fantasy story), their response to the Apple II story was enthusiastic. They worked as a group supplying the adjectives and other words requested by the program. However, the first time through the story, they had no idea how these words were going to be used. The resulting narrative was a humorous, if not entirely accurate, account of what a microcomputer might do over the summer on a deserted college campus. The second time through the program the students chose their words more carefully to fit into the structure of the story they remembered. While this second experience resulted in a new version of the tale, in some ways it was not as entertaining as the first version. In resulting class discussions the notion of appearance versus substance, "how said" versus "what said," and the role of the creator and his audience in giving meaning to a creation were all related to this computer/human-generated story.

3. The Apple II Pops

The Apple II microcomputer can make music-like sounds through the Apple's built-in speaker. A number of commercial programs are available to create these songs with minimal programming. Using the FORTE Music Interpreter by Rainbow Computing Inc., I showed the students how to program and save songs they composed using the Apple's speaker. I wanted the students to construct a nonverbal creation, to become more comfortable with using the microcomputer as a tool, and to learn an introductory programming approach.

At the class concert during which each aspiring composer introduced his creative effort, it became readily apparent that my objectives were achieved. Although the variety of songs (from birdlike melodies to atonal experimentations to K-Tel music advertisements) again showed the diversity in the class, all students completed the assignment and expressed their growing ease with the Apple. They commented on the computer as most forgiving, but also most demanding. Mistakes were easy to correct by retyping the note or line, but putting commands in the correct syntax was required for the program to work. In addition, the students expressed an appreciation of each other's efforts regardless of previous musical experience.

4. Aphorism Construction

Adapting an Aphorism Generator published by J.D. Robertson (*Creative Computing*, August 1979), I programmed the Apple to construct aphorisms such as "Alcohol is the liver of stress" through random combinations of lists of the three key nouns. The class was asked to generate sayings using the format "___ is the ___ of ___" as well. Four students were chosen to do this individually, teams of two and three students were chosen to do it collectively, and the remaining two students worked as a team to select those aphorisms generated by the computer which they thought most creative. After each individual or team had chosen their best two sayings, these aphorisms were read to the rest of the class without identifying the origin. Each class member separately ranked the three best aphorisms; these rankings were then tallied and the highest rated sayings identified. My objective was to stimulate a discussion of individual, group, and random efforts in creating original products.

Although the results were certainly not definitive (one pair had written both of the top ranked sayings), they did lead to a lively debate on the difference between random replacement and meaningful combinations. The idea of "creativity is in the eye of the beholder" was also reinforced by this exercise. The writer *and* the reader put meaning in aphorisms and other creative products.

IMPLICATIONS

The use of these microcomputer activities added greatly to the overall achievement of the course goals. One reason for this success was the close fit between the course topic and the capabilities of the computer. Indeed, creativity with its "battle against fixed attitudes" stresses a concern with new images and experiences. Torrance (1979), a major figure in creativity research, has even recommended a three-stage instructional model to enhance incubation and creative thinking. Activities using a microcomputer can be implemented at each stage in his model. Stage 1 is aimed at heightening anticipation; it tries to motivate learners to relate each learning task to meaningful experiences. The course syllabus and summer story certainly attracted the students' attention and stimulated their curiosity and imagination. Other computer activities such as role playing games and simulations should also warm-up students and increase their anticipation. Stage 2 turns this initial interest into deepening expectation. Song development is one example of the various information processing patterns which Torrance suggests. Stage 3 simply tries to take creative thinking and keep it going. Seeing the various capabilities of a microcomputer in this course should permit students a better understanding of future technological applications in their own lives.

Although psychologists have perhaps focused more on the instructional demands of creativity and problem solving courses than other subjects, the psychological principles underlying such instructional planning and design can be generalized to other college subjects as well. Treffinger and Huber (1975) have emphasized that the content of any course can be analyzed according to an instructional systems approach. Such an approach focuses on identifying specific instructional objectives; diagnosing characteristics of entering learners; developing sequential, learning hierarchies and teaching strategies to accomplish the objectives; and assessing performance to determine learner achievement. Following such a basic instructional model will allow any teacher to estimate the value of particular educational technologies in a chosen course.

The specific issues for incorporating the computer into this current educational technology have been widely discussed during the past decade. Usually the chief criticisms against such a move have revolved around cost and teacher resistance (Trow, 1977). Yet these concerns have mostly been associated with computer-assisted instruction using a large time-sharing system. Recently, there has been a shift from CAI programs aimed at individualized teaching of a single subject matter toward computer-managed instruction in which the system directs the entire instructional process (Splittgerber, 1979). Even with this move toward CMI, there still remains an implementation problem into the public schools. Economic, technological, and instructional design considerations have kept the potential benefits from being realized.

But a revolution is in the making! With the recent rise in accessibility of microcomputers, the cost argument has been greatly deflated. With the growth in microcomputer design and capabilities, many of the technological concerns about access, memory storage, and program languages have also been met. With the growing commitment to instructional design relevant to CAI and CMI, better software is becoming available. Nevertheless, if teacher resistance is not overcome, all these changes will probably be for nought.

The experiences detailed in this paper reveal one possible strategy to introduce the applicability and benefits of microcomputers. By incorporating course content with microcomputer capabilities, students can gradually come to appreciate the vast educational potential in this new technology. With such a focus on computer-enriched instruction, the teacher is less likely to fear displacement from the teaching/learning process. The emphases of CEI experiences are on stimulating class discussions and group interactions, not taking over the teacher's role. As Eisele (1979) has pointed out, the possible uses of microcomputers in the classroom are many: as tools for creative problem solving, games and simulations, drill and practice; testing and test construction. By taking advantage of these uses to enrich classroom activities, today's educator can be accountable and productive without feeling depersonalized or replaceable.

And isn't that a reasonable way to win the battle against restrictive fixed attitudes?

SUMMARY AND CONCLUSIONS

Since many students and teachers are hesitant to use the computer because they are unfamiliar with what it can -- and cannot -- do, it is imperative that ways be developed to introduce gradually a more realistic appraisal of the computer's capabilities. If these benefits are provided through computer-enriched activities which do not reduce the main role of the teacher, more acceptance of such activities should be forthcoming. With the advent of microcomputer systems almost any small college can afford the computing power that was available only to large institutions just a decade ago. An example of such an application in a college course on creativity was provided to illustrate the feasibility of such a CEI approach.

REFERENCES

Eisele, J.E. "Classroom Use of Microcomputers." Educational Technology. October 1979, p. 13-15.

Raudsepp, E. Creative Growth Games. New York: Jove Publications, 1977.

Splittgerber, F.L. "Computer-based Instruction: A Revolution in the Making?" Educational Technology. January 1979, p. 20-26.

Torrance, E.P. "An Instructional Model for Enhancing Incubation." Journal of Creative Behavior. 1979, 13, p. 23-35.

Treffinger, D.J. and Huber, J.R. "Designing Instruction in Creative Problem Solving." Journal of Creative Behavior. 1975, 9, p. 260-266.

Trow, W.C. "Educational Technology and the Computer." Educational Technology. December 1977, p. 18-21.

GIVING ADVICE WITH A COMPUTER

James W. Garson
Department of Philosophy
University of Notre Dame
Notre Dame, Indiana 46556
(219) 283-6471

Over the last three years, my colleague Paul Mellema and I have been at work on an interactive computer program called EMIL which we use to help teach our courses in formal logic. Since June of 1979, we have been devoting our efforts (thanks to funding from The National Science Foundation) to implementing a computerized "copilot" for EMIL which the students can call on to obtain advice on how to tackle problems that they find too difficult to handle on their own.

The methods we are using to provide our students with advice are easy to implement and quite effective. They represent an approach to computerized education that doesn't fit neatly into the standard categories (i.e. record keeping, multiple choice CAI, testing, games, simulations, etc.), but one which has the potential for widespread application. The goal of this kind of tutoring program is to prompt the student to develop and use creative strategies solve problems which do not necessarily have any one correct answer. The advice-giving program is Socratic; it asks the students leading questions concerning the problem before them, questions which help them analyze and resolve their difficulties.

A typical course in formal logic requires students to find formal proofs. Students are presented with a set of formulas of logic and asked to apply certain logical rules to them to derive some other formula. The derivation which they are asked to construct consists of a sequence of formulas each of which follows from previous members of the sequence by one of the rules, and which ends with the formula they are to prove. This kind of learning is important not only in that it provides the foundation for the mastery of the basic concepts of logic, but also because it gives the students the opportunity to learn some of the practical problems and strategies involved in creative thinking, particularly in creative thinking characteristic of mathematics and the sciences.

Giving students practice in the creative solution of formal problems particularly important in science education. Too often scientific knowledge is presented as if it is descended from heaven, or required some form of superhuman intelligence for its discovery. Very little attention is payed in science education to allowing students to appreciate the thinking processes which go into the analysis and solution of scientific problems in a real setting. There is a tendency to obscure the very human process of fumbling around, of trying out strategies, of assessing failures, and of creating better lines of attack, which are all part of the scientists' daily life. A course in logic gives students the opportunity to refine their skills at problem solving in an environment where the difficulty of the problems they confront can easily be adjusted to their growing abilities.

In the standard sort of course, students' abilities at finding proofs vary widely, so that those who do not have an initial knack are severely penalized. Even when strategies for proof-finding are carefully discussed in class, some students invariably complain that they can't do a new problem on their own in spite of "understanding" the lectures. Part of the problem for these students is that they cannot convert a verbal explanation of techniques into a flexible tool for dealing with a new situation.

With a bit of tutoring, most students with these difficulties improve rapidly. If students think out loud while attempting a proof, a gentle nudge here and there often leads to success. If they don't understand the rules, or simply haven't bothered to learn them, guiding them through a proof or two tends to straighten things out fairly quickly and to improve their confidence and motivation. Just as in teaching most skills, the effective strategies involve letting the student perform the task under guidance; lecturing on the proper procedure and telling students to go home and do likewise is relatively ineffective.

Of course there are good reasons why tutoring is not widely used in an introductory course in logic. These classes are usually quite large, so tutoring simply takes too much of the teacher's time. Not only that, grading exercises in proof-finding is tedious, so teachers tend to give students relatively few exercises that require them to create a proof. Even if students can learn on their own, they simply don't get enough practice to develop any skill, unless they catch on right off the bat. Very often, the teacher relies on exercises that require a single answer, such as those that ask the student to fill in the justifications for the lines of a proof that is already completed. This process does familiarize students with the rules, but it gives them no practice in the art of finding a proof.

Computers make it possible to simulate the tutoring situation. Students can enter their proofs at the terminal, and the computer can be programmed to see whether each line of the student's solution follows from previous lines and to describe the difficulty if anything goes wrong. When the student gets lost the computer can make suggestions about how to proceed.

This use of computers in education is particularly interesting because it departs radically from the multiple-choice form which has become almost paradigmatic of computerized teaching materials. A proof-checking program does not require the student to come to any preselected answer, but to find a solution by any of a potentially infinite number of lines of attack. In a sense, the program does not demand an answer, but simply provides an ongoing check of the student's progress in achieving a result. It does not require a set response so much as provide a tool that students can use in their own way to develop a skill.

Compared to multiple-choice programs, proof-checking programs make heavy demands on the computer, the teacher, and the student. The computer must interpret the student's "moves" at the terminal, determine whether they are correct, and respond in an intelligent way. The student and teacher must familiarize themselves with the procedures for operating the computer program, and must put up with the inconveniences caused by having to use a computer which is generally overburdened already, and which occasionally malfunctions. They must also put up with the inevitable mistakes a programmer makes in designing the logic teaching system. And yet, if we are ever to develop computer teaching systems that provide students with tools for learning, rather than merely with ongoing multiple choice examinations, we must overcome these difficulties. Working out effective strategies for proof-checking programs can pave the way for developing less authoritarian styles of computerized education in other areas.

Our logic tutoring program, called EMIL, has several advantages over other programs of the same kind. First, there are a large number of logic text books, each with its own version of the rules of logic. EMIL is the only program that lets teachers supply the program with the set of rules to be used with their textbook, instead of forcing them to use the book which goes with a set of rules written into the program. Second, EMIL is extremely gentle with the student's input, and generally repairs typing mistakes rather than complaining about them. This is important because our students are, for the most part, unfamiliar both with the typewriter keyboard and the notation of logic. Third, the program lets students enter lines at the bottom of the proof which they hope to derive later so that they can work the proof backwards if they like. The reason we allow, and in fact encourage working backwards, is that effective proof-finding strategies require an analysis not only of the formulas already derived, but of the formula to be proven as well. Often the proof-finding problem can be considerably simplified by using the goal formula as a guide-post for determining what the steps previous to it must look like in the completed proof. Our program allows students to record the results of using such strategies right at the terminal, instead of submitting a polished product to the computer. The fourth advantage of our program is the main topic of this paper. Since September of 1979 EMIL has been giving students good advice about how to solve problems that they are unable to do on their own. In this way it is providing a good portion of what can be offered by a human logic tutor.

There are three main strategies for designing a computer program that can offer advice on proof finding. The first is simply to store a completed version of the proof that the student is working on and to store a list of comments that are intended to help a student who asks for aid on computing a particular line. If the comment the student receives proves unhelpful, the student can ask to see the next line of the stored proof, or indeed any number of lines, up to and including the entire proof.

This hint strategy requires that a completed proof must be stored in the computer, along with appropriate comments, for every problem students will work on. It also presupposes that there is likely to be only one reasonable sequence of steps that leads to the conclusion. If students approach a problem in an unusual way, there may not be enough similarity between their proof and the stored proof for the computer to be of any help. Finally, it presupposes a top-to-bottom pattern of proof construction. But frequently the very next steps in a proof will not reveal a strategy that leads to success; such strategies must rather be explained with reference to what happens much later in the proof. This sort of hint routine does not help students appreciate the global strategies which require knowledge not just of where the proof has been, but also of where *it is going*, and these are generally the most useful strategies.

Another technique is to write a program that allows the computer to generate a solution to the student's problem and to recognize standard situations during the course of that solution. This strategy eliminates the need for storing a proof, with commentary, for each problem to be attempted, since the computer generates its own solutions. But this strategy runs the risk of generating strange proofs, which students are unlikely to recapitulate. Also, each formulation of the rules of logic will require its own custom-tailored program for generating proofs. Furthermore, the program to generate comments must be very carefully written to avoid misleading advice. Worst of all, this strategy still does not help students to see global strategies; like the stored-proof strategy, this strategy uses a top-to-bottom approach to proof construction and confines itself to giving advice about the very next line of the proof.

Another difficulty with both of these approaches to the design for an advice-giver is that the program does not attempt to construct advice on the basis of whatever progress the student may have already made on the problem. The failure to build advice around the students partial successes tends to discourage invention of novel, yet promising, partial solutions, to devalue the students' own creative abilities, and to lower their self-confidence. It dampens the students' engagement in the problem-solving process while reinforcing them for stereotyped solutions.

The third approach to the design for an advice-giver, the one we have adopted, overcomes these problems by paying more attention to the techniques actually used by human logic tutors. One of the main things a human tutor should do is provide students with effective problem solving tools for analyzing the situation they are in and for breaking the problem into simpler subproblems to which the same tools can be applied all over again. An effective tutor does not give the solution, or even pieces of it, but instead provides an apprenticeship in the art of asking the relevant questions, the answers of which will lead the student to see how the problem can be broken down into more manageable parts. Questions like "Can you apply this rule to lines you have already derived?" and "What rule could be used to derive a formula of this shape?" when presented in a coherent sequence are very effective in helping students develop strategies which they can learn to use effectively in a wide variety of proof-finding problems.

The actual program that we use to give advice was written by me in about four hours. The implementation was so easy because the advice program does little more than ask students a leading question and then branch to a new question on the basis of their answer. Eventually, the program runs out of questions to ask, and specific advice is given on the basis of the information provided in the student's answers to the questions. The questions can be thought of as being structured in a tree, with the path along the branches being determined by the students' answers, and the advice for each situation being located at the tip of each branch.

Since programming our advice-giver was so simple, the main focus of our attention has been on the creation of a file of questions which have real pedagogical merit. Since the questions are not written into the structure of our program, modifying the question tree in response to what we learn about effective advice has been a painless process which does not require any programming expertise.

Our question file has a very simple format. (See Figure I.) Each record contains the text of a question followed by a list of accepted answers, each followed by a number which indicates which record to jump to in case the student responds with that answer. The last item in each record begins begins with a "*" (which indicates that there are no more accepted answers) and contains text which is printed in case the student does not respond with one of the accepted answers. Most of the questions we ask are answered with "yes" or "no", but we found the use of other sorts of answers more convenient for certain questions. The text of the advice to be given is simply stored in the question file followed by "*." This "*" indicates to the program that this "question" has no accepted answers, and so the program should stop the advice giving process after printing it.

We have built a number of improvements into this simple program. First, the sequence of the questions should vary depending on how much the student has learned and how difficult the problem is. Our first solution to this problem has been to assign each problem that the student is to work on a level number and to use this number to route the question-asking program to separate question trees for each level we have defined

The second enhancement is motivated by the fact that we want to mention items in our questions that change during the execution of the program, for example, the last line number the student has finished in the proof, or the name of the rule that he intends to work with. Obviously the text of the questions in the file cannot mention specific line numbers or rule names. Our solution is to introduce variables, or fill-ins, to our question text, that are then replaced with the corresponding specific information just before the question is printed at the terminal. We have adopted a convention that words beginning with "&" are variables, and so, for example, a line of advice on our question file might read like this: 'YOU SHOULD APPLY &RULE TO LINE &GNUM'. This directs the program to fill in the specific information about the rule name and line number so that the student sees, for example, YOU SHOULD APPLY &OUT TO LINE 5.

FIGURE I: SAMPLE RECORDS FROM A QUESTION FILE

```
1.  'CAN YOU APPLY MP TO ANY PROVEN LINES' 'Y' 2
    'N' 3' *ANSWER YES OR NO'
2.  'APPLY MP TO THESE LINES' '*'
3.  'WHAT IS THE MAIN CONNECTIVE OF YOUR GOAL
    FORMULA?'  '&' 4 'V' 5 '->' 6 '*PLEASE ANSWER
    &, V OR  ->'
```

It may surprise you to learn that although our advice-giving program was running with these two enhancements in September 1979, we were still working on a central portion of the advice giving program in January 1980. That was because we still had to program the most important improvement: the development of subroutines which can answer all the questions which are posed to students by the advice-giver, and comment on any errors in the students' responses. Though students are generally quite accurate in their responses to questions posed by our advice-giver, they occasionally make mistakes that can result in their receiving bad advice. But informing students of their errors is not the only reason for giving the computer the ability to monitor the correctness of the students' responses. Once students run the advice-giver a number of times, they become bored at having to answer a number of pointless questions. The questions become pointless not because they aren't needed in analyzing proof-finding problems in general, but because the student is well aware that a particular portion of the analysis is not needed for the problem being dealt with. When the computer is capable of answering questions itself, we can decide which questions, at which levels of difficulty, should be printed at the terminal, and which we should let the computer answer for itself by examining the proof the student is working on. Experienced students may resent being asked any questions at all and may prefer an advice-giver that merely prints a specific piece of advice. However, we believe that for most students who need the advice-giver in the first place, posing the relevant sequence of questions is much more valuable to their learning problem-solving skills than is their obtaining advice.

At this writing we have a version of EMIL that answers for itself all the questions we pose save one, and we have a method of indicating in our question file which questions are to be asked under which circumstances. We still need to do a lot more research on how obtrusive the advice-giver ought to be as a function of the students' progress and cognitive style. However, the main advantage of our program is that we have complete flexibility over the circumstances under which the program types out the questions.

There is a final reason for programming the computer so that it can answer all the questions. When this is done the program can traverse the question tree on its own and come up with the relevant advice. Once advice is available, the program can follow it to construct proofs on its own. Judging from extensive paper and pencil tests, our advice tree turns out to be highly effective (though not totally effective) in solving logic proof-finding problems. As a result, it is capable of solving for itself the vast majority of problems we give our students. This provides us with an important tool for improving our program. By running a large number of problems through our advice-giver, we can determine the circumstances under which it is unable to do a proof, and then use that information to create a more sophisticated version of our question data files.

Our approach to giving computerized advice has a wide range of applications. It can be used, for example, to help college students with their physics homework, or with tracking down the identity of unknowns in qualitative chemistry, to help medical students learn diagnosis, or even to help people to determine what is wrong with their car, or whether to itemize their deductions. All it takes is a simple program to run the questions and a question file that is carefully constructed to reflect the best strategies that people actually use to solve the kinds of problems which are at issue. Depending on the context of its use, some or all of the enhancements to the basic program that we have developed could be used.

It is worth pointing out exactly how our advice-giving program differs from the standard approach to CAI using the multiple-choice format. The differences are not particularly striking from the programmer's point of view. In both cases, the program is designed to ask questions and to select new questions on the basis of the students' answers. The advice-giving programs generally require a more elaborate branching structure, and they may differ in being unable to evaluate the students' response. But the important differences are the ones that are obvious to the educator, for they have to do with the educational purposes of the program.

Multiple choice CAI attempts to get the student to memorize the correct answers to a certain kind of question. The stress is almost entirely on ensuring that the student learns certain facts. In the case of advice-giving programs, the answers are not part of what is being taught. If anything, it is the questions we would like the student to master. By exposing students, over and over again, to a sequence of questions that have been proven effective in problem analysis, the student learns to develop efficient strategies that can be used over a wide range of problems of a similar kind. Furthermore, the whole process of learning to adopt principles of problem analysis and decomposition is a valuable exercise of problem solving skills that can be applied to virtually any domain where creative thinking is required.

Although advice-giving programs may not look very different from multiple choice courseware to the programmer, they have radically different educational goals, the most important of which is the development of problem solving ability. Given the simplicity of the programming effort as compared to games and simulations, advice-giving programs are particularly attractive for any educator interested in developing the student's creativity.

FROM A THEORY OF READING TO PRACTICE VIA THE COMPUTER

Dale M. Johnson*
Center for Educational Research & Evaluation
The University of Tulsa
Tulsa, Oklahoma 74104

R. Scott Baldwin
University of Miami
Coral Gables, Florida

INTRODUCTION

The purpose of the present paper is to describe a project which uses the unique and high-speed capabilities of the computer to match adolescents with books which the students are *capable* of reading and which they *enjoy* reading. The ultimate mission of the project is to increase the amount students read, which in turn would enable them to become better readers. Enhanced reading ability and attitude would then motivate students to read more; thus, the probability of a cycle of life-long reading habits for the individual would be increased (Mathewson, 1976).

Traditionally, reading has been viewed as a hierarchical skills arrangement. This conceptualization has been reflected in reading instructional programs which incorporate highly structured approaches to word recognition and comprehension skills. The underlying assumption has been that students who are thoroughly grounded in word attack skills, phonics, sight vocabulary, word analysis, spelling, and other reading subskills would use these skills in reading, thereby becoming able readers.

Computer assistance in building reading subskills is not unusual. Illustrative projects in CMI include the Wisconsin System of Instructional Management (WIS-SIM), the Instructional Management System (IMS), the Interactive Training System (ITS), and the Stanford Project (Splittgerber, 1979). Numerous CAI projects in reading have been implemented since the mid-1960s that capitalize on both drill and practice and tutorial modes (Atkinson, 1968; Atkinson, Fletcher, Chetin, & Stauffer, 1970; Atkinson & Paulson, 1970; Madachy & Miller, 1976; and Morrison, 1979). Computers have also been used to establish the "readability" levels of print material (Barry, 1979; Barry & Stevenson, 1975; Fang, 1968; Karten, 1978; and Walker & Boillot, 1979).

Recently, reading specialists have posited that fluent reading develops as an integrated process rather than as a loose collection of reading subskills. Fluent reading entails much more than the sum of specific skills. Such skills provide a necessary but insufficient foundation for the development of mature reading (Goodman, 1976; Smith, 1971). Profound in its implications, the basic idea reduces to a simple formula--young people need to practice reading in order to become good readers (Allington, 1977; Daniels, 1971; Fader & McNeil, 1968; and Squire, 1973). However, students who are in most need of reading practice tend to read the least. The results are that the reading deficit accrues with age, ultimately resulting in secondary students who can neither read on a level congruent with their abilities nor enjoy reading.

RATIONALE

The basis for the project described in this paper is that students do not read partly because they fail to find reading materials which are comprehensible and interesting to them. However, a variety of books do exist that span the abilities and interests of virtually all students. Therefore, a sound argument can be advanced that the problem is not one of availability but one of accessibility.

On the basis of current trends in reading, it could be reasonably hypothesized that if various student characteristics and preferences could be matched with corresponding book characteristics, the probability of the student increasing his or her interaction with print would be enhanced. Further, the increased reading would result in a better likelihood of the student becoming an improved reader and developing more favorable reading habits and attitudes. Therefore, a major task becomes one of maximizing the match between student characteristics and pertinent print characteristics in order to select a book that the student can read and will enjoy. Gilliland (1972) presents this task as follows:

> Readability is primarily concerned with a basic problem familiar to all people who choose books for their own use, or who choose books for others to use. This is the problem of matching. On the one hand, there is a collection of individuals with given interests and reading skills. On the other hand, there is a range of books and other reading materials, differing widely in content, style, and complexity. The extent to which the books can be read with profit will be determined largely by the way in which the two sides are matched... (p. 12).

The time-consuming task of matching students and books has typically relied on published bibliographies, human recall, and trial and error. Such techniques are obviously limited. Finding books which contain all or most of a desired set of characteristics is a complicated task; in fact, it is virtually impossible if the set of characteristics is large and the resources for finding them are limited to human intervention techniques. Simpson and Soares (1965), Jongsma (1972), and Stanek (1975) have found that librarians and teachers, regardless of their intentions, fail to accurately consider the reading interests of young people when purchasing or recommending books.

Research over the past decade or so provides clues as to what variables are important for matching considerations. Robinson and Weintraub (1973) and Squire (1973) have confirmed the importance of reading interests of students when selecting reading material. Further, the reader's age, grade level, socioeconomic background, sex, and cognitive reading ability will tend to influence reading preferences (Ashly, 1970; Carlsen, 1967; Jungeblut & Coleman, 1965; Hansen, 1973; Scharf, 1973; and Soares & Simpson, 1967). Book characteristics which tend to be influential include amounts of dialogue, concreteness of language, type of narration, and the degree of action and conflict provided (Carlsen, 1967; Jungeblut & Coleman, 1965; and Simpson & Soares, 1965). The reader's age, grade, sex, ethnic background, personal history, hobbies, and the book's length, theme, linguistic readability, etc., will combine in a predictable manner to determine just how interesting a particular book will be to a particular reader (Emans & Patyk, 1967; Smith & Johnson, 1972). Thus, it becomes clear that the quality of the match between a reader and a book is a function of numerous personal traits and textual characteristics; however, the literature does provide a comprehensive list of the most important variables.

THE BOOKMATCH SYSTEM

In keeping with the goal of getting the most suitable books to students, the most pertinent variables were incorporated into algorithms which were subsequently coded for computer processing. First, individual student preferences regarding print characteristics are considered. Figure 1 shows the variables on which students can express their personal sentiments.

Three of the characteristics relate to the characters portrayed in the book, one relates to the setting or location of the story, and then 88 topical interest areas are considered. With respect to the "interest areas," the student simply indicated how well he or she likes each particular area from the list of 88 topics on a three choice response format coded as follows: (Y) Yes, very much, (S) Sometimes. It's OK., (N) No, I don't. Of the 88 interest areas, 20 are human drama themes (ecology, drugs, personal beliefs, loneliness, etc.).

Corresponding to each connecting line shown in Figure 1, an algorithm was developed and coded into a FORTRAN subroutine for a main program. However, Figure 2 displays additional algorithms for refining the match between students and books. Figure 2 shows three student traits that represent somewhat different constructs from the student variables (preferences) shown in Figure 1.

The student traits which are linked to various book variables are reading ability, attitude toward reading, and grade level. For example, three algorithms use a measure of student reading ability together with: (1) linguistic difficulty of the book, (2) when the story appeal begins, and (3) the length of the book. Conceptually, better readers can more readily comprehend more difficult text, they can comfortably tolerate more introductory background prior to the beginning of the story appeal, and they will tend to cope better with longer books than poorer readers.

Students' attitudes toward reading are also used (Figure 2) to help discriminate among books. The algorithms use a measured attitude score based on the following heuristics. Students having better attitudes toward reading, as opposed to those with poor attitudes, will be able to cope with longer books, will not require early story appeal, and will rely less on physical action as a prerequisite for enjoying a book. Finally, the grade level, since it is associated with chronological age and maturity, is also compared to the linguistic difficulty (which is measured in grade equivalents) of the books and to the length of the books. Generally, students in lower grades will not enjoy longer and/or more difficult books.

A FORTRAN main program was written for the Xerox Sigma 6 system which includes subroutines incorporating the schemes shown in Figure 1 and Figure 2. The function of the program is to receive individual student variables as input and match them with each of the approximately 5,000 books in the data base. Each variable of each book is compared with the respective student variables and differential weights are assigned according to the degree of match and the relative importance of the variable. The weights are converted to points and summed, resulting in each book being scored for each individual student. The books with the highest number of total points are listed as a personalized bibliography for each student.

The functional operation of the matching process (referred to as Bookmatch) is shown in Figure 3. As can be seen, the first function is to extract a subset of books which are accessible to the student from the book data base. In most cases, this subset is the school's library holdings which have been furnished by the librarian. Second, each variable of each book is compared with either the student's preferences and/or traits. The algorithms accumulate a point total (score) for each book. Finally, the books receiving the highest score (most accumulated points) are printed. This output includes the author(s), title, type of book (fiction, nonfiction, biography, etc.), and a brief annotation of the book.

Each student receives a printed list of his or her own personalized bibliography. The bibliography includes the top 20 books and must contain some nonfiction and biographies along with fiction. Summary reports for teachers, librarians, and administrators are also provided.

DATA BASE

Of significance to the success of the system is the data base which contains information on books written for early adolescent readers (grades 5 - 9). New volumes are periodically being added to the data base which presently contains about 5,000 volumes established over a three-year period. Initially, pertinent book variables were identified through a review of literature in the field of reading. The original pool of variables was verified by a committee of public school reading teachers and university reading specialists. Two of the original variables were subsequently omitted from the list due to variations in printing styles across separate editions of many of the books. These two variables were "number of pictures" (or non-print figures) and "size of type."

Second, a master list containing the titles of books to be included was compiled from published recommendations from widely recognized sources including the American Library Association, Library of Congress, School Library Journal, American Guidance Services, International Reading Association, and the National Council of Teachers of English.

Reviewers were trained to evaluate characteristics of books and to prepare annotations. These reviewers consisted of reading professionals with masters degrees and with either adolescent reading or library experience. The book evaluations were checked for reliability and were further validated by a reading specialist. The annotations were refined by a professional editor. Book data were then coded and stored for computer match of book characteristics with student traits and preferences.

SUMMARY

Based on the notion that practice in reading will be facilitated when readers and appropriate literature are brought together, a project called Bookmatch is presently being implemented. In order to match reader characteristics and preferences with textual characteristics of books, a medium with capabilities for vast data storage, retrieval, and rapid arithmetic comparison operations was required. The computer was the logical choice to handle the task. During the Bookmatch processing, books are scored according to the degree to which they favorably compare to student preferences and characteristics. On the basis of these scores, the computer provides a printout with a personalized annotated bibliography containing the most comprehensible and interesting books that are accessible to the individual student.

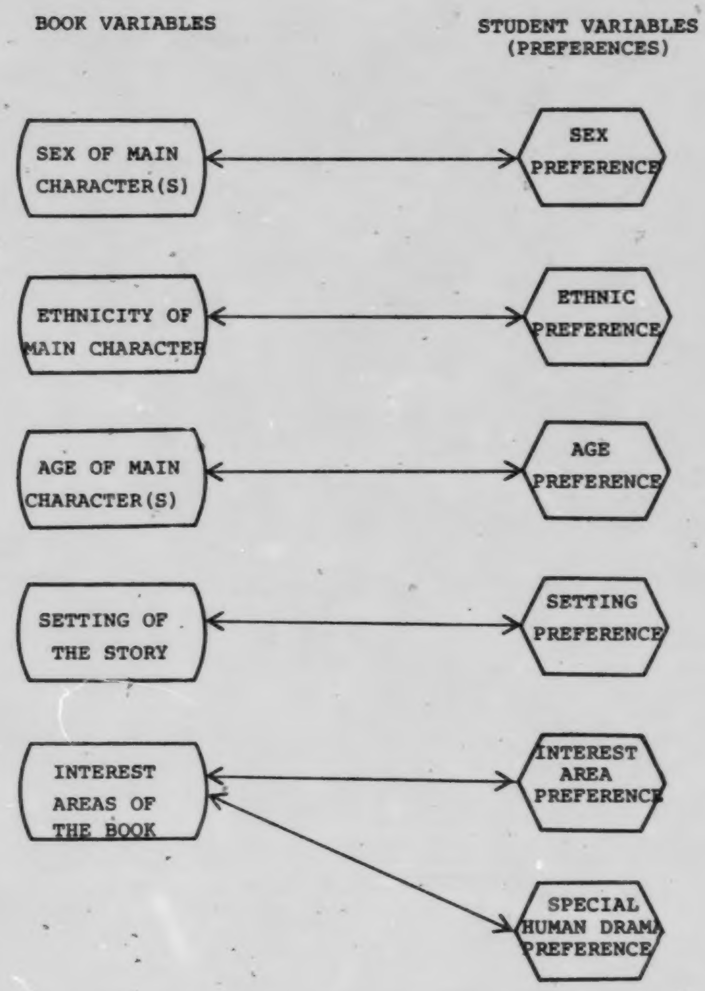

BOOK VARIABLES
STUDENT VARIABLES (PREFERENCES)
SEX OF MAIN CHARACTER(S)
SEX PREFERENCE
ETHNICITY OF MAIN CHARACTER
ETHNIC PREFERENCE
AGE OF MAIN CHARACTER(S)
AGE PREFERENCE
SETTING OF THE STORY
SETTING PREFERENCE
INTEREST AREAS OF THE BOOK
INTEREST AREA PREFERENCE
SPECIAL HUMAN DRAMA PREFERENCE

Figure 1. Schematic for matching algorithms of student preferences and book variables.

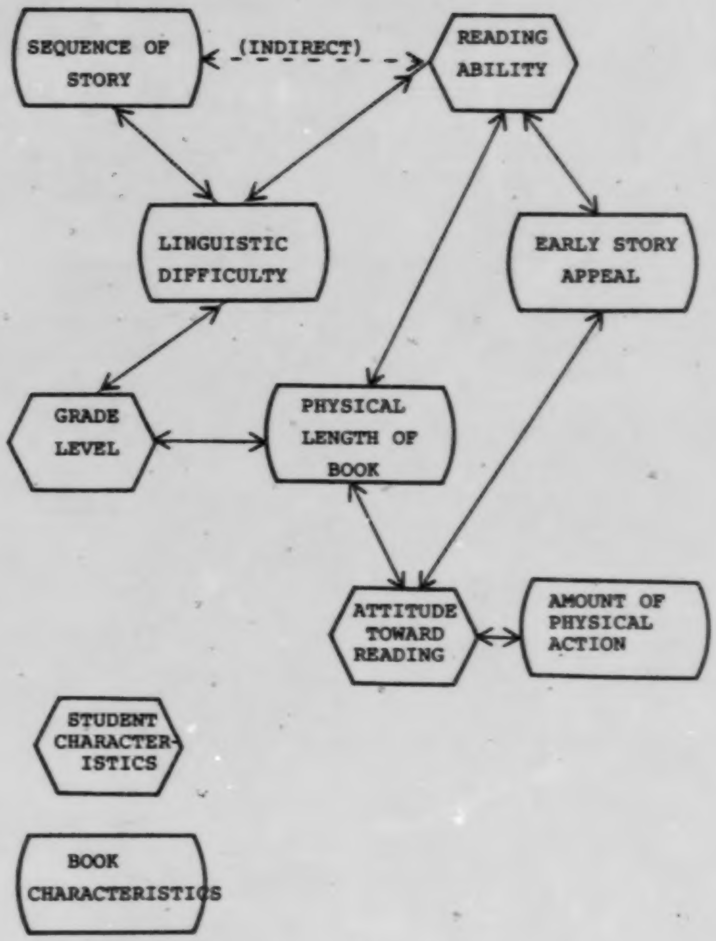

SEQUENCE OF STORY
(INDIRECT)
READING ABILITY
LINGUISTIC DIFFICULTY
EARLY STORY APPEAL
GRADE LEVEL
PHYSICAL LENGTH OF BOOK
ATTITUDE TOWARD READING
AMOUNT OF PHYSICAL ACTION
STUDENT CHARACTER-ISTICS
BOOK CHARACTERISTICS

Figure 2. Schematic network for matching algorithms of student traits (characteristics) and book variables.

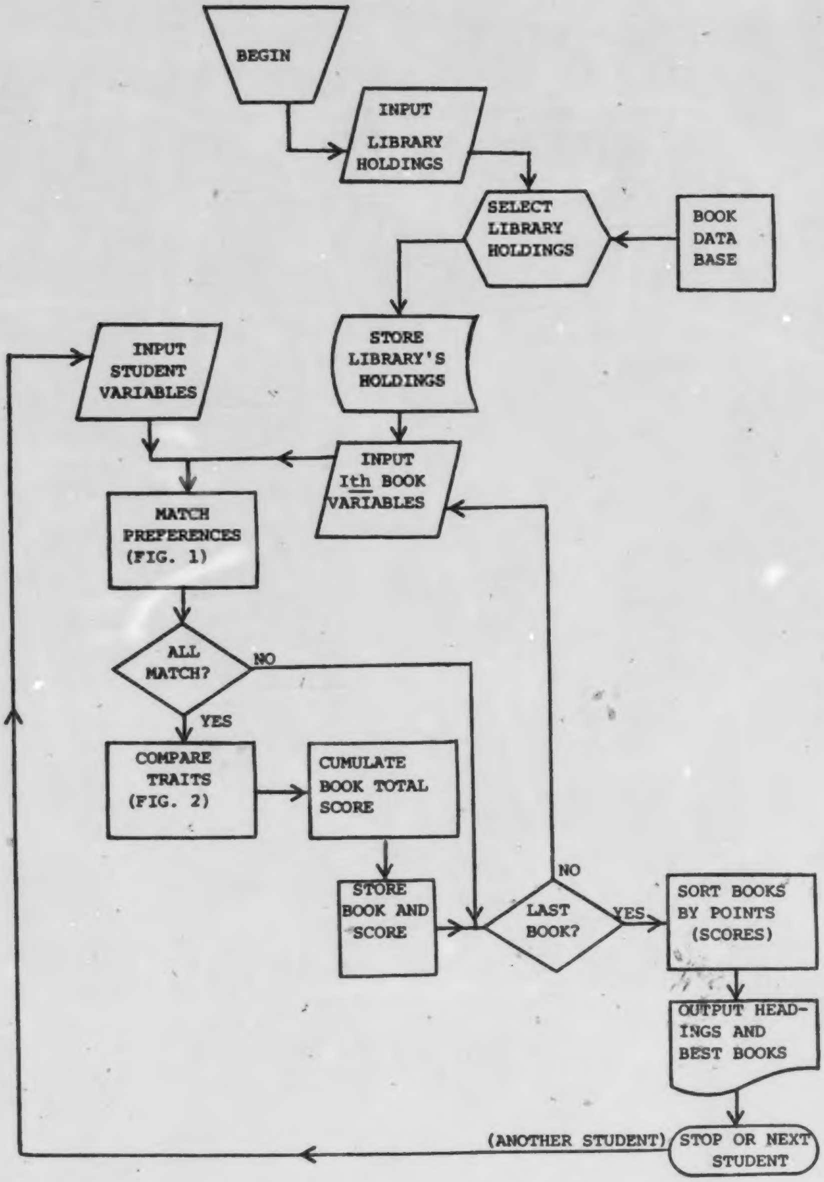

BEGIN
INPUT LIBRARY HOLDINGS
SELECT LIBRARY HOLDINGS
BOOK DATA BASE
STORE LIBRARY'S HOLDINGS
INPUT STUDENT VARIABLES
INPUT Ith BOOK VARIABLES
MATCH PREFERENCES (FIG. 1)
ALL MATCH?
NO
YES
COMPARE TRAITS (FIG. 2)
CUMULATE BOOK TOTAL SCORE
STORE BOOK AND SCORE
LAST BOOK?
NO
YES
SORT BOOKS BY POINTS (SCORES)
OUTPUT HEADINGS AND BEST BOOKS
(ANOTHER STUDENT)
STOP OR NEXT STUDENT

Figure 3. Functional Schematic for BOOKMATCH operation.

REFERENCES

Allington, R.L. "If They Don't Read Much, How They Ever Gonna Get Good?" Journal of Reading, 1977, 21, 57-61.

Ashly, L.F. "Children's Reading Interests and Individualized Reading." Elementary English, 1970, 47, 1088-96.

Atkinson, R.C. "Computerized Instruction and the Learning Process." American Psychologist, 1968, 23, 225-39.

Atkinson, R.C., Fletcher, J.D., Chetin, J.C. & Stauffer, C.M. Instruction in Initial Reading Under Computer Control: The Stanford Project. Technical Report 158. Stanford, Calif.: Institute for Mathematical Sciences in the Social Sciences, 1970.

Atkinson, R.C. & Paulson, J.A. An Approach to the Psychology of Instruction. Technical Report 157. Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, 1970.

Barry, J.C. "Computerized Readability Levels--Their Need and Use." Journal of Educational Data Processing, 1979, 16, 10-15.

Barry, J.C. & Stevenson, T. "Using a Computer to Calculate the Dale-Chall Formula." Journal of Reading, 1975, 19, 218-22.

Carlsen, G.R. Books and the Teen-age Reader. New York: Harper & Row, 1967.

Daniels, S. How 2 Gerbils 20 Goldfish 200 Games 2,000 Books and I Taught Them How to Read. Philadelphia: The Westminster Press, 1971.

Emans, R. & Patyk, G. "Who Do High School Students Read?" Journal of Reading, 1967, 10, 300-4.

Fader, D.N. & McNeil. Hooked on Books: Program and Proof, New York: G.P. Putnam's Sons, 1968.

Fang, I.E. "By Computer: Flesch's Reading Ease Score and a Syllable Counter." Behavioral Science, 1968, 13, 249-51.

Gilliland, J. Readability. London: University of London Press, 1972.

Goodman, K.S. "Reading: A Psycholinguistic Guessing Game." In Singer and R.B. Ruddell (eds.). Theoretical Models and Processes of Reading. Newark, Delaware: International Reading Association, 1976.

Hansen, H.S. "The Home Literary Environment--A Follow-up Report." Elementary English, 1973, 50, 97-8.

Jongsma, E.A. "The Difficulty of Children's Books: Librarians Judgments Versus Formula Estimates." Elementary English, 1972, 49, 20-6.

Jungeblut, A. & Coleman, J.H. "Reading Content That Interests Seventh, Eighth, and Ninth Grade Students." The Journal of Educational Research, 1965, 58, 393-401.

Karten, H.A. " G M Program Rates Level of Text Difficulty." Computerworld, 1978, (May), 9.

Madachy, J. & Miller D.J. The Use of CAI in the Language Program at Gallaudet. Santa Barbara: Association for the Development of Computer-Based Instructional Systems, January, 1976.

Mathewson, G.C. "The Function of Attitude in the Reading Process." In H. Singer and R.B. Ruddell (eds.), Theoretical Models and Processes of Reading (2nd ed.), Newark, Delaware: International Reading Association, 1976, 665-76.

Morrison, F. TICCIT. Dynamic Phoenix, 1976 (July), 45-6

Robinson, H.M. & Weintraub, S. "Research Related to Children's Interests and to Developmental Values of Reading." Library Trends, 1973, 22, 81-108.

Scharf, A.G. "Who Likes What in High School." Journal of Reading, 1973, 16, 604-7.

Simpson, R.H. & Soares, A. "Best-and-Least-Liked Short Stories in Junior High School." English Journal, 1965, 54, 108-11.

Smith, F. Understanding Reading. New York: Holt, Rinehart and Winston, Inc., 1971.

Smith, J.R. & Johnson F.N. "The Popularity of Children's Fiction as a Function of Reading Ease and Related Factors." The Journal of Educational Research, 1972, 65, 397-400.

Soares, A.T. & Simpson, R.H. "Interest in Recreational Reading in Junior High

School Students." Journal of Reading, 1967, 11, 14-21.

Splittgerber, F.L. "Computer-Based Instruction: A Resolution in the Making?" Educational Technology, 1979, 19, 20-5.

Squire, J.R. "What Does Research Reveal About Attitudes Toward Reading?" In R.A. Meade and R.C. Smith (eds.), Literature for Adolescents: Selection and Use. Columbus, Ohio: Charles E. Merrill, 1973.

Stanek, L.W. "Real People, Real Books: About YA Readers." Top of the News, 1975, 31, 417-27.

Walker, N. & Boillot, M. "A Computerized Reading Level Analysis." Educational Technology, 1979, 19, 47-9.

*Note
Dale Johnson will present paper.

NON-HARMONY:
A VITAL ELEMENT OF EAR-TRAINING IN MUSIC CAI

Joan C. Groom-Thornton
and
Antoinette Tracy Corbet
School of Music
North Texas State University
Denton, TX 76203
(817) 788-2791 ext. 269

INTRODUCTION AND BACKGROUND

Computer-assisted instruction (CAI) in ear-training has become an important assistant to classroom teaching in the first two years of college-level music theory. The student's aural skills (which are needed to deal with the three basic elements of music -- harmony, melody, and rhythm) can be greatly reinforced with such individual instruction. Part of the success of this reinforcement is that while these elements are integral parts of a complex multi-dimensional subject, they can be separated from each other to a great extent, so that classroom teaching (and CAI lessons) can concentrate on virtually one element at a time. In particular, this system allows the less advanced student to concentrate his efforts upon developing his understanding of one broad subject at a time. As his skills and confidence grow, he is better capable of identifying these elements in various combinations, and eventually of understanding them in their musical context.

As soon as the student has developed some skill in each of these three areas and has begun to understand the integrated whole, it is necessary to introduce the concept of "non-harmony." As the name implies, this element lies outside the realm of the established harmonic structure but relies on melodic and rhythmic context to differentiate among the various forms that it takes. Despite its rather negative title, non-harmony is a very positive part of music -- an aspect of the concept of variety which helps to delineate style. In practice, non-harmony takes the form of short melodic and rhythmic patterns which augment the harmonic context. These patterns fall into approximately ten categories for which terminology is fairly standard. Each term refers to a type of non-harmonic tone which defines its own special pattern and context. Although each pattern is rather specific, melodic and rhythmic context can vary somewhat and harmonic context can vary greatly. For this reason, the necessity of drill and practice of such variance presents a logistics problem in the classroom but is greatly helped by random access and tranposition available with CAI.

METHOD

The ear-training CAI system at North Texas State University employs the Automatic Music System (AMUS) designed by Professor Dan W. Scott of the Computer Sciences Department. The AMUS system consists of a Motorola 6800 microprocessor with a CRT terminal and specialized music hardware (MUSOR). The microprocessor translates the musical score and accesses the MUSOR, which is the actual sound-producing apparatus engineered by Professor Scott. The AMUS system is connected to an HP 2000-F Timeshared BASIC computer, which provides additional file storage capabilities. This system uses a flexible score language for notation input and offers a variety of timbre and articulation possibilities. Of primary importance is the facility of rapid playback which is a substantial advantage over other computer-based music systems in its cost range.

The non-harmonic tone lessons were constructed using a simple melodic and rhythmic four-voice chorale style. This simplicity was not due to any system limitations, as the system is capable of rapid and complex articulations of rhythm, a wide range in melody, and up to seven simultaneous voices for harmony. Rather, the simplicity was maintained to limit the number of variables of elements other than those being tested and to give a fairly normal musical context. For each example heard, the basic context was a five-to seven-chord progression of four voices in quarter-notes. To this strictly harmonic setting, one or more non-harmonic tones were added in eighth-note motion. Since there were ten categories to be covered, the following factors determined the order in which the terms were presented. Group I consisted of the five non-harmonics that normally occur off the beat or in a rhythmically weak position as compared to the note of resolution which comes directly after. Group II consisted of the four non-harmonics that occur on the beat or in a rhythmically strong position, and usually displace or delay a note of the harmony. The last category normally has no such rhythmic definition. Due to the rhythmic contexts, Group I is considered easier to hear than Group II. Also, in actual practice in music literature, the patterns in Group I generally occur more often than those in Group II.

Within Group I, the five unaccented non-harmonics (with their standard abbreviations) were presented in this order:

1. unaccented passing tone (UPT)
2. neighboring tone, upper and lower (UN, LN) also sometimes called "auxiliaries"
3. anticipation (ANT)
4. escape tone or echapée (ET)
5. changing tones (CT)

Within Group II the four accented non-harmonics were given this order:

6. accented passing tone (APT)
7. suspension (SUS)
8. retardation (RET)
9. appoggiatura (APP)

The final example was:

10. pedal tone (PED)

The internal order of the two groups was chosen to present the most-often heard categories first, in order to start the student with the most familiar patterns. The only exception to this practice occurred with categories 8 and 9. The less-often heard retardation (#8) was placed before the appoggiatura (#9) in order to present it directly after the suspension, on which it is patterned. Also, this order was selected because summary lessons are used in various places between individual presentations, and a summary to show similarities and differences between the suspension and retardation would logically come before proceeding to the next (and less-related) non-harmonic.

The general need for both immediate and cumulative summaries of such a large number of elements resulted in seven such lessons being inserted into the sequence already discussed. While each of the ten single-element lessons seemed well served by the examples (which were carefully written to provide maximum exposure to the variations encountered in music), the summary lessons gradually grew in number of examples since they demonstrate a variety of combinations of these elements. Of course, this variety is further expanded by random selection and transposition.

A standard format for the student's answer was kept for all of the lessons. Because of the complex situation presented by non-harmony, the student's response could not be reduced below three elements: the rhythmic, melodic, and non-harmonic contexts. (This sequence best represented the student's understanding of the factors involved, as determined by the author's experience in teaching.)

For the first element, the student types a number from one to six, locating the beat with which the non-harmonic is associated. (Although many examples are seven chords long, a basic precept of non-harmony is that it must finally resolve into harmony. Therefore, non-harmony could not be associated with the seventh beat.) The second element of the student's answer is a capital "S" or "B" which indicates that the non-harmonic occurred in the soprano or bass melodic voice. (The choice was limited to the upper and lower voices to afford a measure of variety while avoiding the unnecessary complication of determining locations in inside voices when these voices can be extremely close.) The third and last element of the student's answer is the abbreviation that identifies the non-harmonic itself from among the ten possible labels. Even though this part of the answer would be obvious in any of the single-element lessons, the act of typing the abbreviation reinforces the student's aural impression and establishes a pattern of response which does not need to be altered for the summary lessons. All elements of the answer were separated by commas, even when the example involved more than one non-harmonic response.

One other factor, a chord tone, was added to a few random examples within each lesson. A chord tone may resemble a non-harmonic tone in rhythmic and melodic placement, but it is part of the harmony (as opposed to non-harmony), and its pattern of usage does not necessarily parallel that of a non-harmonic tone. This factor was added to insure that the student would truly listen for non-harmony and not just attempt to label anything that moved! These chord tones do not require any response from the student other than to understand that they were notes which were not part of the non-harmonic answer.

RESULTS

In final sequence, there are seventeen lessons, delineated as follows:

N1. UPT - Unaccented passing tones occur most often of any of the non-harmonics and are located equally well in soprano or bass.

N2. UN, LN - Upper and lower neighbors occur mainly in soprano, as reflected in the examples.

N3. ANT - Anticipations occur almost exclusively in the soprano and usually very close to the end of the progression.

N4. Immediate summary of Lessons 1, 2, and 3 (the most-often heard unaccented non-harmonics).

N5. ET - Escape tones occur mainly in the soprano voice.

N6. CT - Changing tones involve a definitive four-note pattern and occur only occasionally in the bass.

N7. Immediate summary of Lessons 5 and 6 (somewhat similar patterns).

N8. Cumulative summary of all unaccented non-harmonics (Lessons 1 through 7).

N9. APT - Accented passing tones, like unaccented ones, occur in both soprano and bass.

N10. Summary of APT and UPT (Lessons 1 and 9). These two patterns occur quite often and are alike except for rhythmic placement.

N11. S - Suspensions occur in both voices in many cases.

N12. RET - Retardations parallel suspensions in every way except that they resolve up instead of down.

N13. Immediate summary of the accented non-harmonics covered thus far (Lessons 9, 11, and 12).

N14. APP - Appoggiaturas may have slightly varied melodic patterns and most often occur in the soprano.

N15. Cumulative summary of all accented non-harmonics (Lessons 9, 11, 12, 13, and 14).

N16. PED - Pedal tones are the least often used, but are the most obvious non-harmonics to identify, occurring mainly in the bass.

N17. Final cumulative summary of all of the non-harmonics (This lesson contains the most examples).

SAMPLE

The following example shows Lesson N1 (UPT) with comments on specific features.

```
C:  N1, 1/13/80, JCGT & ATC
C:  CHORD PROGRESSIONS WITH UNACCENTED
C:  PASSING-TONES
H:  FIRST ANSWER IS THE NUMBER OF THE
H:  CHORD AFTER WHICH THE NON-HARMONIC
H:  TONE SOUNDS.
H:  SECOND ANSWER IS "S" OR "B" FOR
H:  SOPRANO OR BASS VOICE WHERE THE NON-
H:  HARMONIC TONE OCCURS.
H:  THIRD ANSWER IS THE ABBREVIATION (UPT)
H:  FOR THE NON-HARMONIC.
H:  TYPE LETTERS IN CAPITALS (WITH SHIFT
H:  KEY).
H:  (SOME MOVING EIGHTH-NOTES MAY BE CHORD
H:  TONES.)
```

The "H" statements are "hints" which the student may branch back to at any time to obtain reminders about valid answers and format.

```
T:  $N, HERE'S A LESSON TO HELP YOU LEARN
T:  NON-HARMONIC TONES.
T:  THESE WILL BE UNACCENTED PASSING-TONES.
P:  2
```

The "T" statements are texts which the student sees, and the "$N" addresses the student by his own name. The "P: 2" is a two-second pause (with the text still on the screen) to allow the student to absorb the information.

```
CS:
S:  2
T:  I'LL PLAY 5 TO 7 CHORDS.
T:  YOU TYPE:  WHICH CHORD (1,2,3,4,5 OR
T:  6) THE NON-HARMONIC OCCURS AFTER,
T:  WHETHER THE NON-HARMONIC IS IN SOPRANO
T:  OR BASS (S OR B) IN CAPITALS.
T:  THE ABBREVIATION FOR THE NON-HARMONIC
T:  (UPT) IN CAPITALS.
T:  TYPE A COMMA (,) AFTER EACH PART OF
T:  YOUR ANSWER.
T:  FOR INSTANCE:  3,B,UPT
T:  MEANS THE NON-HARMONIC OCCURRED AFTER
T:  THE THIRD CHORD, IN THE BASS VOICE,
T:  AND IT WAS AN UNACCENTED PASSING-
T:  TONE.
P:  10
```

The"CS" then clears the screen so that the following text can appear as a stationary group without rotating off the screen. "S: 2" double-spaces the text of added visual impact. The text contains specific instruction on the anwer format and gives a hypothetical student response with explanation. Again, there is a pause of ten seconds for comprehension.

```
T:  HERE'S AN EXAMPLE (WITH ANSWER), $N--
T:  $R 4CD 3C 2E GE
T:  4EE
T:  4G 3BD 2D GQ
T:  4E 3CU 2G CUQ
T:  4F 3C 2F ADQ
T:  4G 3GD 2D BE
T:  CUE
T:  4GD 3B 2F DQ
T:  4CU 3G 2E CH
```

```
T:  ?
P:  20
T:  %S %E
T:  YOUR ANSWER:  5,S,UPT -- RIGHT, $N ?
P:  10
```

The next section plays a sample example for the student (lines 145 - 205 process the sound) and then gives the correct answer with a pause for comprehension.

```
T:  REMEMBER:
T:  TYPE A "," BETWEEN THE PARTS OF YOUR
T:  ANSWER.
T:  TYPE A "?" TO RE-HEAR THE MUSIC.
T:  TYPE " //HINT" FOR A HINT ABOUT
T:  ANSWERS.
T:  (YOU MAY WANT TO USE PAPER AND PENCIL
T:  TO JOT DOWN ANSWERS BEFORE YOU TYPE
T:  THEM IN.)
P:  10
```

The student then sees the final reminders on basic procedures for dealing with the musical examples.

```
T:  LET'S GET STARTED, $N!
V:  123456SBUPT,
Q:  10<R\D\P4/5\3 >
```

The actual examples are then announced to the student. "V" gives the valid characters for the student's answer. For every lesson the first eight characters are the same; the numbers 1 through 6 are the possible answers to the first question of beat (rhythmic placement); and "S" and "B" are the possible answers for voice (melodic) placement. The last characters label the non-harmonic and range from a single one- to three- letter grouping (for the single-element lessons) to the complete list of all non-harmonic possibilities for N17. In this sample lesson of N1, only the single abbreviation of UPT is valid. Finally, "Q" formats the questions. In this case there are 10 possible examples from which the program makes a random selection and a direct comparison. The student completes the lesson when he answers four out of five times correctly, and he is allowed three tries before the correct answer is given by the computer.

Ten examples then follow which are similar to the sample example of lines 145 through 190.

DISCUSSION

The seventeen lessons of this non-harmony project are all presented in parallel format to establish a consistent pattern of thought and approach and to minimize confusion as the lessons gradually become more complex. Each element is introduced in a lesson devoted solely to its problems and contexts, and summary lessons gradually combine the elements in a logical order. Data on students' responses are kept and made available to the classroom teacher for evaluation and advising.

SUMMARY AND CONCLUSIONS

Music poses interesting instructional problems because it shares with other fields (for example, chemistry and math) a well-defined vocabulary for musical elements. However, musical teaching must include showing the relationships among these elements. Non-harmony is an essential component of these relationships, and thus forms a model for the solution to the problem of teaching relationships in many fields.

SELECTED REFERENCES

Apel, Willi. *The Harvard Dictionary of Music*. Harvard University Press, Cambridge, 1969.

Bales, W. Kenton and Joan C. Groom. "AMUS: A Score Language for Computer-Assisted Applications in Music." Presented at the 1979 Association for Educational Data Systems Convention, Detroit, Michigan.

Hamilton, Richard L., and Dan W. Scott. "A New Approach to Computer-Assisted Instruction in Music Theory." Presented at the 1977 Conference on Computers in the Undergraduate Curriculum, Denver, Colorado.

Hofstetter, Fred T. "Computer-Based Recognition of Perceptual Patterns in Harmonic Dictation Exercises." Presented at the 1978 Association for the Development of Computer Instruction Systems Conference, Dallas, Texas.

Killam, Rosemary N., W. Kenton Bales, Richard L. Hamilton and Dan W. Scott. "AMUS: The Computer in Music Instruction." Presented at the 1979 Texas Music Educator's Association Conference, Fort Worth, Texas.

Ray, Douglas, and Rosemary N. Killam. "Melodic Perception Development and Measurement Through CAI." Published in the proceedings of the 1979 National Educational Computing Conference, Iowa City, Iowa.

Computer Literacy

A CASE FOR INFORMATION LITERACY

Dr. B. B. Schimming
IBM Corporation
10401 Fernwood Road
Bethesda, MD 20034
301/897-2090

INTRODUCTION

Although digital computers have been present in educational institutions for over 25 years, only a small percentage of students have been exposed to them. As the cost of computing continues to decline, it is becoming financially feasible to introduce the majority of students to computing and data processing.

Dr. Andrew R. Molnar (1) is an articulate spokesman for the expanded integration of the computer into American education. Citing our shift from an industrial society to a knowledge-based society together with the international challenge to our technological leadership, he makes a compelling case for computer literacy. Dr. Molnar is not alone in his views. The term "computer literacy" is being increasingly used and discussed at educational conferences and in the educational literature.

The objective of this paper is to explore the future content of computer literacy education. Based upon an assessment of current computer/data processing education, trends in technology, and a projection of the future information environment, a set of three future skill categories is defined.

- Computer/Data Processing Professionals (2) - the computer experts
- Information System Professionals (3,4) - the integrators of the computer and business processes
- The End Users - the information system customer

The first two categories are, of course, key areas requiring educational planning and investment. However, from a statistical point of view, they will represent a small minority of the population interacting with future information systems. It is the end user majority for which computer literacy is probed in this paper. The result of this exercise is a recommendation for the creation of a course entitled "An Introductory Course in Information Literacy" for the majority of undergraduate students who are pursuing a major other than computing.

TECHNOLOGY DIRECTIONS

The 1980s will conceivably see an order of magnitude growth in computing power. Responsible for this growth will be the continued decline in cost of electronic components together with the elastic demand for data processing services. The concept of the computer will change as processing capability is distributed into many devices. Today's hobbyist with a microprocessor at home capable of communicating with a distant large computer is a beginning example of distributed processing. One of the reasons for this trend is that the cost of computing has dropped below the cost of communication. Thus it becomes desirable to satisfy local data processing requirements at the source while communicating only that data required by other sites. Communication costs will also continue to decline as newer forms of broadband communication, such as satellite transmission, become commonplace. Again, as in the computer industry, change will be dramatic in the communications industry. There can be little doubt that information technology growth will continue to be significant through the latter part of this century.

NOTE: The views expressed in this paper are those of the author and do not necessarily represent the views of the International Business Machines Corporation.

The key implication with regard to educational curricula is that not only will computers change, but the information environment in which they operate will change; the way data elements are moved, stored, and used will be altered as well as the way they are processed.

An analogy to transportation systems can be drawn here. Vehicles, roadways, and parking facilities are components of a transportation system just as computers, communications, data, and applications are components of an information system. Vehicles are not studied in isolation by urban planners, and correspondingly computers should not be studied in isolation from other components of the information system.

CURRENT COMPUTER/DATA PROCESSING EDUCATION

For the purpose of this discussion, it is convenient to think of three classes of students that correspond to the skill categories presented in the introduction.

- Computer Majors - those preparing for careers in data processing.
- Information Systems Majors - those preparing for careers in management with a major in information systems.
- Other - those with career objectives other than computing or data processing . . . the future end users.

Dr. J. Hamblen's most recent inventory of computing in American education (5) provides insight into the relative distribution of these classes: in excess of 90% of the students fell into the "other" category while 30% of the institutions offered courses in the first two categories. From these statistics, it would seem the computer major is receiving more than his fair share of resources.

The status of information systems (IS) education is addressed in a recent article (6) reporting preliminary findings of the ACM Curriculum Committee for Information Systems. Prof. Jay F. Nunamaker, chairman of the committee, stated: "The IS field integrates systems analysts, statistics, management science, accounting, economics, finance, marketing, production and computer and communication technologies . . . "The U.S. has nearly five computer science departments for every IS department according to a recent study not connected with the ACM investigation. Nevertheless, the nation has a much higher demand for personnel, such as IS graduates, who have a combination of technical and organizational skills than for computer science graduates with 'solely' technical skills."

With the previous discussion as a basis, certain conclusions can be made regarding future educational requirements of the three categories of students previously defined.

FUTURE EDUCATIONAL REQUIREMENTS

- Vocational and Computer Science Professionals - The demand for this type of professional together with the curriculum needed to prepare him or her appears to be relatively well understood and served by the educational community.
- Information System Professionals - As reflected by the Nunamaker study, the information system concept has been slower to emerge. However, due to the efforts of the ACM (3,4) and the nation's business schools (42 of the 52 satisfactory IS undergraduate programs were components of business or management colleges) (6), the problem is understood and undoubtedly steps will be taken to improve the quality and quantity of information system graduates.
- All Others (the future end users) - It is this group for which current computer literacy appears to be inadequate. They will take their places in marketing, production and administration with a cursory knowledge of the computer but not the information system in which they function. A hard-earned lesson in the computer industry is that a major cause of information systems failure has been that they didn't do what the users expected. The end users must represent their needs as input to the information system design process.

The following course description addresses the minimum elements of an information literacy offering.

AN INTRODUCTORY COURSE IN INFORMATION LITERACY

There are four key components in this information literacy proposal: computers, application programming, data, and communications. The computer portion could be satisfied with the traditional "Introduction to Computing/Data Processing" course currently available at many institutions. There is no need to elaborate here on the contents of this type of course.

Application Programming

Coding in a high-level language is not sufficient to understand the role of application programming. In fact, for the generalist, it may be of secondary importance to an awareness of the application requirements/preliminary design process. The following topics should be addressed:

- Requirements Study
  - Documenting information flow in an organization
- Preliminary Application System Design
  - Transaction definitions
  - Screen formats
- Generation of a Software Specification
- Design of an Application Software Module
  - Flowcharts
  - HIPO diagrams
- Coding of an Application Software Module
  - Coding
  - Testing
  - Debugging

Data

Much as the treasurer is responsible for the money resource in an organization, the business community has come to realize the value of their data records and the need to assign management responsibility for their safekeeping and use. An information literacy course should include the following subjects.

- The Concept of Data as a Resource
- Data Characteristics
  - Physical relationships
  - Logical relationships
- Storage Alternatives
- The Role of Data Base Administration
  - Data dictionary concepts

Communications

The fourth and final component of the information literacy offering, communications, should impart an awareness of the regulatory, cost, and technological aspects of the links in a contemporary information system.

- Industry Structure
  - Common carrier services
    - Switched
    - Leased
    - Value added - packet switched
    - Satellite
    - Other
  - Tariffs
- Information Networks
  - Combining communications and data processing
  - Network architectures
    - Centralized
    - Distributed
- Design Tradeoffs
  - High volume - batch
  - Interactive
  - Public vs private

It should be noted that each of these four topics is a complex specialty in its own right. No pretense is made that skill in any of the topics would be developed through the course, but rather a conceptual knowledge of the elements in an information system would be gained. Also of equal importance, the role of the end user in an informtion system design would be established.

A two-semester sequence would be desirable; however, a one-semester survey course could also be of value. The business school, because of its precedence in information systems curricula, would be a logical choice to offer this introductory course.

In conclusion, it is hoped that this presentation stimulates thinking in the educational community with regard to its role in preparing students to coexist with information technology in the 21st century.

NOTES

(1) Andrew R. Molnar, "The Next Great Crisis in American Education: Computer Literacy," AEDS Journal, Association for Educational Data Systems, Volume 12, No. 1, Fall 1978, p. 21.

(2) "Curriculum '78: Recommendations for the Undergraduate Program in Computer Science," Communications of the ACM, Vol. 22, No. 3, March 1979.

(3) "Curriculum Recommendations for Undergraduate Programs in Information Systems," Communications of the ACM, Vol. 16, No. 12, December 1973.

(4) R. L. Ashenhurst, ed., "Curriculum Recommendations for Graduate Pro-

fessional Programs in Information Systems," a report of the ACM Curriculum Committee on Computer Education for Management, 1972.

(5) J. Hamblen, "Computer Education in Higher Education -- Status, Alternatives and Needs," AFIPS, 1978.

(6) "ACM Cites Dearth of DP Programs," Computerworld, November 5, 1979.

A BYTE OF BASIC

Judith A. Hopper

Arapahoe County
District 6
Grant Junior High
Littleton, Colorado 80122
(303) 795-2560

INTRODUCTION

The average U. S. citizen has not the foggiest idea of how computers work and how pervasive their influence actually is. Consequently, he has no idea of what to do when a computer makes a mistake; he has no idea of how to vote on local, state, or national issues involving computers (e.g., the establishment of a national data bank); he is, in short, culturally disadvantaged.

It is therefore essential that our educational system be modified in such a way that every student (i.e., every prospective citizen) become acquainted with the nature of computers and the current and potential roles which they play in our society. It is probably too late to do much about adults, but it would be disastrous to neglect the next generation.

- Committee on Computer Education, Conference Board of the Mathematical Sciences

For today's children, understanding computer fundamentals is one very important factor in building an informed citizenry for the future. Students who are computer literate will have better career opportunities and less career shock. They will be better able to cope in a world with rapidly moving and ever-changing technology. Thus, in this era called the Age of Information, schools can no longer delay in bringing the computer into their curricula.

A BYTE OF BASIC

Seeing the need to get some kind of program with computers started in her school district, the author of this paper wrote a proposal to pilot a computer literacy and BASIC programming class at the junior high level. The district purchased an Apple II microcomputer and things were off and running.

A class of fifteen eighth and ninth graders was chosen for the project. The class, to be one semester long (eighteen weeks), met five days a week for fifty-five minutes. The Apple II is a 32K system with a Centronics printer. It was felt a hard copy would be a benefit in that the students would have something tangible to take home to show to their parents.

The class was begun by acquainting the students with computers in general: what is a computer system, what are its parts, and what does each part do. Using their knowledge and expanding what they knew, the class discussed various types of input/output devices as well as particular usage, i.e., in business, industry, science, and research.

In order to get the students on the computer as quickly as possible, flowcharting and algorithms were the next topics presented. As it was important for the students to be able to break down a problem into its component parts and analyze each step of the solution, flowcharting was introduced as a pictorial representation of this process.

Simple programming was introduced next to acquaint students with the computer. Feeling comfortable with the computer and learning the on-off procedures in using the computer were the main objectives. The introductory programming was simple to insure each student initial success at the terminal.

Egg cartons were then used to build personal computers for each student. In beginning to write more difficult programs the students had to put their program through their "EGG 12" system first to be sure it was doing what they had intended. Once it was demonstrated that it would run on their computer, they could run it on the Apple. Using this technique forced them to evaluate their own work and make corrections as needed.

As the class progressed through the semester, more and complex programming was introduced. Most all of the common BASIC statements were used (LET, IF-THEN, TAB, GO TO, REM, FOR-NEXT, PRINT, etc.) and also some library functions (SQR, ABS, INT, RND). In addition, as the machine being used has color graphic and string variable capabilities, these were soon added to the repertoire of the student programmers.

Interspersed among the programming lessons, other topics were presented. In answer to the question of what does the computer do with the program once it has been typed in at the terminal, machine language and the binary system were discussed. Changing numbers back and forth from base two to base ten, and adding, subtracting, and multiplying in base two were some typical class activities. A comparison was also made of a program written first in BASIC, then in assembler, and finally in machine language.

The concept of language interaction resulted in the introduction of the origins of BASIC and the history of computing. Beginning with Stonehenge, the abacus, and the slide rule, the class proceeded through Pascal, Babbage, Hollerith, Aiken, Hopper, von Neumann, and others. This discussion of the development of the computer and its changing faces served to illustrate where we are in terms of both current computing capabilities and conjectures about the future.

The class enjoyed learning to use various computing devices, the abacus being of particular interest to them. The slide rule and calculator were also introduced as well as a ten-year old Hewlett-Packard 2000 series computer, which required marking Hollerith cards with a pencil. As the students were exposed to these devices, they learned both the limitations and the capabilities of each.

An important part of any literacy class is not only learning what a computer can and cannot do but also keeping up with some of the literature available on computers. Throughout the course the students had to read a number of articles from various periodicals acquainting them with what was going on in the world of computing. They had to bring current articles in from magazines and newspapers and write reports. The vocabulary of computer-ese was carefully discussed so that as complete an understanding as possible was available. Any new developments of computer usage were also pointed out and discussed.

Field trips were made to observe the use of computers outside the classroom, and several people working in the computing field were invited to speak to the class. They discussed various jobs directly related to using the computer, the training required, the responsibilities that go with the jobs, and the availability of openings in the field.

It was hoped that with the completion of this course the students would have a working background knowledge of programming, that all fears of a computer would have been extinguished, and that they would be aware that computers have great potential and capabilities but also limitations. If the development of the computer continues at the rapid pace at which it is now moving, the students' world will truly be a computerized world. Any knowledge of the field they can gain now will serve to make their lives that much easier upon completion of their education.

FINDINGS

This first class was an enthusiastic group of fifteen students. They were excited about working with the computer and could hardly wait for some hands-on experience. This enthusiasm continued throughout the semester. Each time the students wrote a program, they immediately wanted to test their programming abilities, gaining a great deal of satisfaction in watching the computer do what they had intended.

But while there was enthusiasm, there was also some apprehension among the students. One of the primary goals of the class was to overcome this fear. The students need to be comfortable and feel at home with the terminal. Providing them some initial success with computers was a way of achieving this goal.

The biggest problem for the class was the availability of only a single terminal. With fifteen students writing two and three programs a week it was difficult to schedule time so they each could use the computer at least twice a week. Some came in before school, others during lunch and after school, as well as during the full hour being used during class. It is hoped this problem will be alleviated next semester with the acquisition of four additional microcomputers.

Another problem encountered was the lack of typing skills in the students. Having to use the one-finger-hunt-and-peck method consumed a lot of time when they were ready to type in their programs. As the class progressed through the semester they became better and faster at getting their programs in, but it remained a time-consuming problem throughout.

Now that the first class is over, the students are asking for a second class at a higher level. They are very interested in the many facets of the machine and would like to pursue its capabilities. Unfortunately a second course in computing is not currently available at the junior high level (it is at the high school), but it is hoped that these students can be used as resource personnel. With their ability to program and run demonstrations, they may be called on by teachers wanting information or simulations. By using the computer in other classrooms, it can become an integral part of the entire school curriculum.

FUTURE OUTLOOK

The computer programming and literacy class at Grant Junior High was a pilot class for the district. Before jumping in with both feet the district felt it best to try an initial program, evaluate that one, and go on from there.

Concurrently with piloting this class, visitations to other school districts with a computer curriculum already in full swing were made. As at Grant, there appeared to be good student interest, but also considerable frustration over the inadequate number of terminals available.

Then, with recommendations from visitations, attendance at several conferences, and research in the area of computer education, a proposal was formulated for the implementation of microcomputers into the school district. The proposal calls for one junior high and one senior high in the district to be fully equipped to begin the implementation. It is hoped that these two schools working together would be able to iron out any problems that would arise so the full district-wide implementation would progress smoothly.

The proposal for the computer implementation specifies definite curriculum recommendations. At the elementary level the gifted and talented program already is doing some programming. The proposal calls for some subtle exposure of the other elementary students -- at least to acquaint them with computers so that their fears are eliminated.

At the junior high level it is planned that all students at some time during their three-year career take a computer literacy course (possibly nine weeks). Structured within the class would be a basic introduction to computing -- what the computer can and cannot do. In addition to the literacy course a class in computer programming would be offered for those students who would like to pursue the programming aspect. Also it is hoped other disciplines (math, science, social studies, and English) would want to use the computer as a demonstration tool for simulations, data processing for experimental or classroom results, tutoring individuals, or any other functions -- limited only by the imagination.

Programming in higher level languages as well as practice in developing software would be offered at the senior high level. The proposal also recommends the computer be used as a tool in higher level math courses, in science and social science classes, and in the business department for data processing.

As staff members at each of the schools, in addition to the students, would need exposure to the machines, the proposal also provides for inservices and actual classes for faculty members not familiar with computers. It is hoped that teachers in various disciplines would then be interested in the computers and want to use them as a demonstration tool in their classrooms.

The school board has just recently accepted this proposal for a two-year implementation of microcomputers into the schools in the district. The hardware requirements are not merely cost-effective but also offer the opportunity to improve the quality of education. Where once this type of technology was used by a few privileged persons, today it is becoming ubiquitous. A degree of flexibility has also been built into the proposal so changes or enhancements to improve the program can be made.

Education is missing an opportunity if not a responsibility if it fails to provide its students with the necessary background to move into the "information age." We need to start now to provide the public with the understanding of this information science and technology. As J.C.R. Licklider put it: "People must master the technology or be mastered by it."

EQUIPMENT LIST

- High Schools:
  - Apple II 48K
  - Video
  - Double disk
  - RF Oscillator
  - Firmware Card
  - Radio Shack 48K
  - 32K addition
  - Double disk
  - 15 Radio Shack 4K Level II
  - Master slave
  - Interfacing connectors
  - Printers where needed
- Junior High Schools:
  - Apple II 48K
  - Double disk
  - Printer
  - 8 Radio Shack 4K Level II
  - Master slave

A COMPUTER WORKSHOP
FOR ELEMENTARY AND
SECONDARY TEACHERS

Herbert L. Dershem
John T. Whittle
Hope College
Holland, Michigan 49423
(616) 392-5111

INTRODUCTION

Although computers have been used by secondary teachers for a long time, only recently has the microcomputer made it economically feasible for the elementary teacher to use the computer in the classroom. In addition, new technological developments have made it possible for secondary teachers outside the fields of mathematics and science to use the computer as a classroom tool. Recent reports and recommendations (1,2) have emphasized these facts and indicated the need to make teachers aware of the potential the computer possesses as a tool in the classroom.

This paper describes a two-week workshop which was offered by the authors in the summer of 1979 to provide teachers with knowledge of computers as they are applicable to the teachers' classes.

ORIGIN AND OBJECTIVES OF THE WORKSHOP

Through working with students and teachers at both the elementary and secondary level, the authors became convinced that the computer is a valuable educational tool. But it was apparent that the computer could be used effectively only if the classroom teacher was aware of its potential. We believed that once the teacher learned to use the computer, he or she would be able to use it in creative and innovative ways in the classroom. The biggest step was overcoming the teachers' fear and awe so that they could implement their own classroom computer applications or adapt those of others to their own needs.

With this in mind, we designed a two-week workshop with the following objectives:

1) The teacher will be sufficiently familiar with the operation of a computer to instruct others in its use.

2) The teacher will know where to find resources for ideas, activities, and programs related to the classroom use of computers.

3) The teacher will know the BASIC language well enough to write simple programs, to introduce the language to students, and to read any BASIC program and make minor modifications to it.

4) The teacher will have sufficient understanding of the way a computer works to explain it to students.

5) The teacher will know the techniques and approaches most frequently used in instructional computer applications and have experience in their use.

6) The teacher will have designed and implemented at least one computer activity for his or her classroom and be capable of developing others.

7) The teacher will know the types of computer equipment available for classroom use and be aware of advantages and disadvantages of each.

The workshop was intended for teachers who had no previous experience with computers and who wished to explore ways in which they could improve their teaching by use of the computer.

WORKSHOP FORMAT

The workshop was divided into two parts, a laboratory and a lecture, each of which met for one hour and fifteen minutes every day for two weeks. Upon completion of the course, the teachers received two hours of college credit.

The BASIC language was taught in the laboratory portion of the course. Actually, two laboratory sessions were held each day, one before and one after the lecture, half of the participants attending each session. Ten Radio Shack Level II 16K TRS-80 computer systems, loaned to the college by Radio Shack, were set up in the laboratory and each participant was seated in front of a unit. The textbook, "Using

BASIC"by Julien Hennefeld (3), was chosen because it contains many sample BASIC programs to illustrate concepts. A set of these programs was placed on cassettes for each laboratory; then during the laboratory each participant would load a program from cassette. Next, the instructor would talk about the concept illustrated, have the participants run the program, and usually ask them to make modifications to see the effect. The topics covered in the laboratory during the first week were as follows:

Day 1 - How to use the computer
LET statements
PRINT statements
Day 2 - GO TO statements
IF-THEN statements
Day 3 - INPUT statements
FOR-NEXT statements
Day 4 - Subscripted variables
Graphics
Day 5 - String variables
PRINT@ statements

During the second week the laboratory time was used for each participant to design and implement a useful classroom computer activity. The instructors assisted the participants in the design and programming of the activities. Additional features of the BASIC language and advanced programming techniques were covered as they were needed. On the final day of the workshop each of the participants presented the results of his or her project to the entire group.

In the lecture portion of the workshop, the participants were divided into elementary and secondary educators. The topics of the lectures for each group follow:

Day 1 - How Computers Work
Day 2 - Techniques and Approaches in the Classroom
Day 3 - Elementary - More techniques and approaches
Secondary - Survey of Resources
Day 4 - Elementary - Survey of Resources
Secondary - Problem Solving Principals
Day 5 - Computer Literacy
Day 6 - Design Considerations for Instructional Hardware
Day 7 - Experience with Instructional Software
Day 8 - Survey of Computer Equipment
Day 9 - Elementary - More experience with instructional software
Secondary - How to Teach Computing
Day 10 - Presentation of Projects

Both laboratories and the lecture were held in the morning. The room containing the computers was also left open in the afternoons so the participants could return and work on their projects. Although this afternoon work was not required, many teachers did take advantage of this opportunity. In addition, several had TRS-80 systems at home or at their schools which they used in the afternoons or evenings.

RESULTS

Twenty educators from three local school districts attended the workshop. Of these, twelve were secondary teachers, four were elementary teachers, and four were administrators. All four administrators attended the first week only. Also, some of the teachers who had previous computer experience took the second week only.

A brief description of the projects developed by the teachers is found in the Appendix. Both the instructors and the participants were amazed at what had been accomplished in a two-week period. An enthusiasm for computer use was generated by the workshop so that participants went back to their schools and pushed for the purchase of a computer for their classroom.

A further indication of the success of the workshop is the demand for it to be repeated this summer. This demand is coming from teachers who saw the effect the workshop had on last year's participants. As a result, the same workshop will be offered twice this summer as well as once during the academic year. Hope College has now established its own microcomputer laboratory, so the computers will not be borrowed systems.

The only modification to the workshop which we plan is not to allow participants to register for just one week. Those who took only the first week missed the important experience of putting together their own software. It was also difficult to adjust the workshop to those who came in during the second week. As a result, the workshop will not serve the needs of those teachers with some previous computer experience.

Interest has been expressed in a workshop specifically for teachers who are already using computers. We are considering offering such a workshop, which would cover advanced programming and software design techniques. In this workshop we would invite each teacher to bring along a student so that they could participate as a teacher-student team. From our observations, much of the software developed is actually done by such teacher-student teams with the teacher doing the design and the student the programming.

REFERENCES

1) Milner, S. "An Analysis of Computer Education Needs for K-12 Teachers." National Educational Computing Conference Proceedings, Iowa City, 1979.

2) Taylor, R., Poirot, J., Powell, J., and Hamblen, J. "Computing Competencies for School Teachers: A Preliminary Projection for All but the Teacher of Computing." National Educational Computing Conference Proceedings, Iowa City, 1979.

3) Hennefeld, J. Using BASIC: An Introduction to Computer Programming. Prindle, Weber & Schmidt, Inc., 1978.

APPENDIX. PROJECTS COMPLETED BY PARTICIPANTS IN THE COMPUTER WORKSHOP

Physics experiment simulation - A falling body experiment is simulated by the computer. The student is asked to provide the appropriate formulas to calculate expected results.

Geometry drill - A drill and practice exercise is conducted using geometric terminology.

Spanish drill - A drill and practice on Spanish grammar and vocabulary is conducted entirely in the Spanish language.

Ordering - This program is intended for use at the elementary level. The student takes a list of items and places them in a described order. Three implemented orderings are alphabetical, fractions, and decimals.

Parts of Speech - The student is given a word list and after picking a word from this list, leads the computer to identifying the word by answering the computer's questions about its part of speech.

Word house - This program is intended for students learning English as a second language. The student is required to place each of a list of words into its proper category and word house.

Presidential drill - This program drills students on presidents and their terms of office.

Golf statistics - This data collection and analysis program simplifies the paper work of the golf coach.

Career counseling aid - A student interested in a career in accounting can sit down with this program and find out what options exist in this field and the education necessary for pursuing each option.

Test generator - This test generation program simplifies the high school math teacher's job of creating examinations by randomly choosing problems which have parameters that may also be randomly selected.

Carrying drill - This program drills the student on carrying skills in multiplication by presenting randomly generated problems. If the student responds incorrectly the program carries out the multiplication, carefully specifying each carry value along the way.

Mortgage payoff - The student can use this program to examine the effects of varying the parameters on mortgage payoffs.

MICROCOMPUTERS
AND COMPUTER LITERACY:
A CASE STUDY

Robert J. Ellison
Hamilton College
Clinton, NY 13323
315-859-4138

I will discuss the role of the microcomputer in computer education at Hamilton College. The focus will be on the kinds of applications I have found useful and on the methodology that was used to develop the systems on the microcomputer. My initial interest in the microcomputer has been its use as an aid to teach computer literacy. I will discuss the development of an interactive statistical package that is the core of a three week introduction to computing and programming. The package permits elementary programming and file management to be introduced without an extensive introduction to programming.

While the microcomputer can be an excellent instructional aid, it quite often is inconvenient for program development. Most of our applications could not be developed on a larger machine, however, since we are very dependent on programming the screen and the graphics displays. I will discuss our experience with the UCSD PASCAL(TM) system which is now available on a number of different computers. It has significantly aided the design and management of the project. The system's extensive use of prompt lines makes it easy for the novice student to use.

It was my goal to develop systems which could be easily modified and expanded. The microcomputer is not large enough to support a universal package; furthermore, the more complete systems usually also bring with them a more complicated control structure. Since our goal is to introduce computing to an audience with little if any computer experience, we wanted a simple and often specialized user interface. But if we are not going to offer a fairly complete package for the micro, then we should design a system which can be easily modified. The statistical package I will discuss was written using a library of procedures which can be integrated into a custom system. The package has been supplemented by both the mathematics and biology departments for their own special applications. Some of the materials will also appear in the development of a file maintenance system in the advanced computer science course on data structures. The limited professional support for academic programming in a small college practically forces us to write general purpose softwear.

THE ENVIRONMENT

Hamilton College is a liberal arts institution with an enrollment of 1600 students and a faculty of about 135. Over 600 students a year will take courses which use the computer. The computer science offerings consist of a two-course programming sequence (6) and two advanced courses which emphasize file structures and topics in data base organization. The first programming course is taken by about one third of the students before they graduate. The academic computing is done using a remote batch entry to the IBM 370/168 at Cornell University. We will shortly be able to use the interactive system at Cornell also. The advanced computer offerings are taught using PASCAL and microcomputers.

I am exploring the possibility of applications which do not require the full capability of the IBM/370 or which can take advantage of the simpler operating environment of the microcomputer. While the Cornell system supports a quite adequate number of statistical packages and programming languages, for the novice the system is still too complex to be easily used. A number of applications involving elementary program instruction or simple statistical analysis might be better served on a smaller machine or one specially configured for the use.

Learning the control language for the use of a computer system is a special problem for those non-computing courses that want to explore the use of the computer as a tool. The focus is too often on the details of the system rather than the problem to be solved. A well designed microcomputer system might be of service here also.

We have selected the TERAK microcomputer as our primary unit. The CPU is a Digital Equipment LSI-11. The TERAK was chosen for its graphics capability, a 320 by 240 raster image, and the wide range of software available. Operating systems for the TERAK include DEC's RT-11, the UCSD PASCAL(TM) operating system, which we use most often, and the Cornell Program Synthesizer (13), which I will discuss below. The TERAK can also be used as a terminal to Cornell, and with the use of a communications package text files can be transferred between the two computers. Programs exist to convert files between the RT-11 and the UCSD disk formats. I have developed software to transfer the graphics and character buffers to a Printronix which has a plot capability.

GOALS

Our most immediate need was for a system to support a new offering for computer literacy in our winter term. Hamilton follows a 4-1-4 calendar. It was my intent to offer a short course which could help meet the demand for computer literacy. Since our first programming course is designed for a general audience, I did not feel the need to teach programming. On the other hand it seemed important to introduce students to the use of the computer to store information and hence to the concept of a file.

I felt it was quite important to involve the student more than as a passive observer of the computer. Our greatest problem was finding programs and applications which involved the student with more than selecting an option from a menu. I am particularly interested in examples which involved the maintenance of files on the computer. An immediate application was the use of a text editor and text formatter. The UCSD screen editor is very easy to use. I had also written the formatter (9) in PASCAL. I added user-defined macros with the output directed to the printer, disk file, or screen. While the text editor could be used to discuss files and updating of information, I wanted a more complex example.

The use of a statistical package seemed a possibility. The choice was in part motivated by use of the SAS system (12) which makes extensive use of both permanent and temporary files. A SAS run consists of a number of procedures to list, sort, and modify the file. Most procedures create a file to pass data to the next step of the process. SAS also supports limited programming in PL/1. In most cases the program code is applied to each record of the file. It seemed that a package such as SAS could serve to introduce the concept of a file, program variables, and elementary programming. Using files as the primary way to pass data is ideally suited for a microcomputer with limited memory.

The use of SAS presented some problems for our short course. Our computing on the Cornell system is primarily done by remote batch entry. Although I preferred to use an interactive system for this course, the IBM control language for the interactive system was too complex for our audience. It also seemed difficult to introduce in such a short time the syntax and semantics of even an elementary part of PL/1.

My solution to the above problem was motivated by the availability of a programming system for the LSI-11 (13) called the Cornell Program Synthesizer. The system includes an editor for the language PL/CS (5), which gives a template for each major construction. For example, a single keyboard command generates the template

```
     If (condition) then
        {statement}.
```

Statements are checked as entered. The system is designed for teaching programming and includes elaborate editing and tracing commands. I wanted to include some of these features to assist in entering elementary programs for the statistical system.

THE STATISTICAL PACKAGE

The system supports real, integer, and character variables. The core of the system is an elementary file management system that includes modules to open and close file, get or put the next record, and retrieve or store individual variables in a record. Each statistical file consists of a data file and a directory file that lists the variable names and types. The file management system is stored in the system library and can be called from any PASCAL program. The system is designed for instructional use, not to support large-scale statistical analysis. It can serve as a tutorial for the use of a more complete, flexible, and faster system such as SAS. The system consists of the following general programs.

Create:

The record and directory files are created from the text files made by the editor.

List:

The directory and the data are listed on either the screen or the printer. Variable headings are automatically printed. A single record may take several lines.

Sort:

The file may be ordered by any of the variables in ascending or descending order. Multiple sort keys can be specified. It may also be used as a procedure in other programs.

Modify:

This module supports an elementary programming language to permit modification of the file. The statements are prompted and errors are noted on entry. The program is saved on the disk for documentation. Currently, an individual statement can be edited as it is entered but cannot be changed after it is accepted. I will not include elaborate editing, since most of the programs are rather short and are easily entered again.

The statements supported and the template provided include:

Assignment:

```
{variable name} = {expression};
```

New variables are declared simply by entering the name. The variable type is automatically set by the type of expression.

Conditionals:

```
If (condition) then do:
    {statements}
    end;

Select;
    when (condition) do;
        {statements}
         end;
    when (condition) do;
        {statements}
         end;
    ...

    otherwise do;
       {statements}
       end;
    end;
```

The SELECT statement has been recently added to the IBM version of PL/1. Only the statements associated with the first true condition will be executed. The OTHERWISE block may be empty. The SELECT was used instead of an IF..ELSE construction as the former seemed to be less confusing. I chose not to implement the GO TO statement. It not only complicated the design of the interpreter for the language but also created the possibility of infinite loops.

Input/output

```
Output (file name);
```

The current record is written to the selected file. When the file is opened the user has the option of specifying which variables will be written. The record has all the variables in the input file as well as any new variables introduced by the program. Currently two output files are permitted. The program prompts for an output file if none is specified.

If an expression or variable is not correctly entered, a list of the variables and their types can be requested before the correction is made. The program is automatically indented to reflect the nesting of conditionals.

The following is a sample program. The variables QUIZ1, QUIZ2, FINAL, and MIDTERM are on the input file. The variable TOTAL is created by this program. The lower case entries were supplied by the program.

```
TOTAL = 0 ;
if NOT(QUIZ1 MISSING)  then do;
   TOTAL = TOTAL + QUIZ1 ;
   end;
if NOT(QUIZ2 MISSING) then do;
   TOTAL = TOTAL + QUIZ2 ;
   end;
TOTAL = TOTAL + FINAL + MIDTERM ;
output  DEMO  ;
```

The system was used for the first time this winter. Our program editing was not as elaborate as apparently was needed, but the overall student response was quite positive. To the student, the basic program unit is the file which for a general course on computer literacy was appropriate. The limited programming also helps explain how a computer works. We currently plan to use the same system with our first programming course to introduce files and interactive computer systems. A variant of the system will be used in the first year biology sequence.

The system is easily expanded. New programs are written in PASCAL with the PASCAL input/output functions replaced by those in the new file management system. It is particularly easy to have a student develop an independent PASCAL program for a specific file and then replace the input/output functions to generate a general purpose program. The system, especially the file management section, has been an excellent source of examples for the advanced computer science students.

DEVELOPMENT

I would like to discuss the way we have managed the development of this system and the use of UCSD implementation of PASCAL. A small college often does not have the programming support available for the development of large systems. The proliferation of a number of specialized packages would only compound the maintenance of these programs. While the initial effort is greater, I have tried to develop general-purpose programs which can serve in other applications, for example, the file management system which supports both the statistical package and general file maintenance programs for an advanced computer science course on file structures. The graphics library, which I have not discussed, is used by mathematics, biology, and physics.

It is clearly desirable to reuse software, but keeping each of the various subsystems supplied with the latest revisions can be difficult to manage on a small system. Programming in PASCAL and the UCSD implementation of it have both assisted in this task. A vital feature of PASCAL is the ability to define new data types. PASCAL includes the simple variable types integer, real, and character. But for the kinds of systems programming involved in this project, we needed more elaborate structures. For example, we had to maintain a directory for each file which maintains the variables and their type. The definition of such a directory had to be consistent throughout the system. In PASCAL we could define a directory in several steps.

```
  Const {compile time constant}
    Namelength = 10;

  Type

  Namestr = packed array[1..Namelength]
    of char;  {string}
  Vardescript = record
      { describes file variable }
    Name :   namestr;
    Position :  integer;
    Typeofvar:  (realtype,integertype,
         chartype);
    Length:  integer { length in bytes}
    Max : real;
    Min : real;
    end
```

We can then declare the type directory as

```
  directory = array[1..maxvars]
    of vardescript;
```

A procedure GETDIR whose purpose is to read the directory could be declared with the following heading:

```
  Procedure GETDIR( VAR D: DIRECTORY);
```

If it is necessary to add more information to our description, such as the number of missing values, then we only need add a line such as

```
  missing : integer;
```

to the original definition. That change is then reflected in all procedures which use that data type.

The UCSD system has included some non-standard features. I have tried to avoid the use of them when possible, but their definition of a unit has been quite valuable. A unit is a separately compiled library of subprocedures. It can also include type definitions. I keep the system type definitions in the library. Those types are then copied into each program which uses that unit. Changes in the common definitions are then reflected in all programs which use that module. It has been an excellent tool to maintain consistency in student written programs.

The procedures and definitions can be included in the program by the simple command

```
  uses {unit name};
```

The statistical system has a unit devoted to file management, another to parsing and evaluating expressions, and a third unit to provide run-time support. In addition we have other units with graphics procedures or random number generators. The use of units does not affect portability of the system. The source code of the procedures could be inserted instead of the uses statement.

SUMMARY

We are continually surprised by the ease with which major changes can be made to the system whose length now exceeds 5000 lines. The use of PASCAL and the UCSD unit has played a major role. It has been very easy to develop the system incrementally and to quickly train students to assist in the programming. PASCAL is well suited to the kind of systems programming involved in this project.

We have been pleased with the student reaction to the statistical system. The combination of that system along with the PL/CS programming system has provided a quick introduction to programming and the use of files for our course on computers and society. We are planning to develop tutorial materials so the system can be used by other courses. Further evaluation of the system will have to wait until it has been used by other disciplines. We have been successful in having other faculty use the modules to develop their own specialized software. We are now evaluating what additional modules should be placed in common units.

REFERENCES

1. Austing, R.H., Barnes, B.H., Bonnette, D.T., Engel, G.L., and Stokes, G.S. "Curriculum '78: Recommendations for the Undergraduate Program in Computer Science--A Report on the ACM Curriculum Committee on Computer Science." Comm. ACM 22, 3(March 1979), 147-166

2. Austing, R.H., and Engel, G.L. "A Computer Science Course for Small Colleges." Comm. ACM 16, 3(March 1973), 139-147

3. Committee on the Undergraduate Program in Mathematics of MAA. A compendium of CUPM recommendations. MAA 11 (1975), 528-570

4. Conway, R. Primer on Disciplined Programming Using PL/CS. Winthrop 1978

5. Conway, R. and Constable, R. "PL/CS--A Disciplined Subset of PL/I." Technical Report 76-293, Department of Computer Science, Cornell 1976

6. Ellison, Robert J. "A Programming Sequence for a Liberal Arts College." Proceedings of the 1980 SIGCSE Symposium on Computer Science Education

7. Grogono, Peter. Programming in Pascal. Addison-Wesley, 1978

8. Jensen, Kathleen and Wirth, Niklaus. Pascal User Manual and Report. 2nd edition. Springer Verlag 1974

9. Kernighan, Brian and Plauger, P.J. Software Tools. Addison-Wesley 1976

10. Lopez, A.A., Raymond, R., and Tardiff, R. "A Survey of Computer Science Offerings in Small Liberal Arts Colleges." Comm. ACM 20, 12(Dec 1977), 902-906

11. Nevison, J.M. "Computing in the Liberal Arts College." Science 194 (Oct. 1976), 396-402

12. SAS Users Guide 1979 Edition. SAS Institute, 1979

13. Teitelbaum, R.T. "The Cornell Program Synthesizer: A Microcomputer Implementation of PL/CS." Technical Report 79-370, Department of Computer Science, Cornell University 1979

14. Van Loan, Charles. "Computer Science and the Liberal Arts Student." Technical Report 79-376, Department of Computer Science, Cornell University 1979

Invited Session

PERSONAL COMPUTING: AN ADVENTURE OF THE MIND.

Paul Hazen
Applied Physics Laboratory
Johns Hopkins University

ABSTRACT

Under grants from the IEEE Computer Society, The Johns Hopkins University, Radio Shack, and other agencies, the International Instructional TV Cooperative, source of instructional TV materials to all educational TV networks nation-wide and internationally, has finished and is marketing the implementation of a six-course national educational TV series aimed at the pre-college level in the area of personal computing and computer literacy. The name of the project is "Personal Computing: An Adventure of the Mind."

The objectives of this new series are to illustrate the uses of personal computing, to demonstrate the interface of humans and machines, to identify the fundamentals of communication in personal computing, and to motivate students to be innovative in their own applications of personal computing. Since the personal computer is viewed by many as a mind multiplier, a further objective of this educational TV series is to greatly increase the number of minds that can be multiplied by taking personal computing to millions of children in classrooms across the country.

Education and informational programs are closely allied in that both attempt to communicate facts, concepts, and ideas. Both need to be designed with specific objectives in mind. Some of the objectives to be discussed are both attitudinal and informational in nature; that is, they deal with feelings as well as facts. The underlying thrust throughout is that . . . LEARNING CAN BE FUN!

Invited Sessions

EDUCATIONAL COMPUTING: PAST, PRESENT, FUTURE

Ronald W. Collins
Dept. of Chemistry
Jefferson Science Bldg.
Eastern Michigan University
Ypsilanti, MI 48197
(313) 487-0106

ABSTRACT

During the past ten years considerable effort has been devoted to optimizing the role of computers in education. A variety of tutorial CAI programs have been written, large data bases of questions for computer-generated exams have been amassed, sophisticated software for graphics has been developed, numerous data reduction and simulation programs have been written, and the use of on-line classroom computing has been studied. In addition, the emergence of minicomputers and microprocessors has lowered the cost of computing. Nevertheless, the overall impact of the computer on education has been minimal. One of the major deterrents has been the poor transportability of programs from one computer to another. As a result, educators are often required to develop their own software for use in courses; however, many instructors have neither the time nor the expertise to do so. Will the new stand-alone home computer systems improve this situation? Possibly, but for most educational uses, the need for programming time and expertise will still be high. Furthermore, the several generations of changes in hardware have not yet led to significant improvements in computer-based pedagogy. Educators simply must give more consideration to the question of how the computer can reshape the way and style in which we teach. To date, most instructional computing has simply been appended to the traditional pedagogical framework, thus encouraging the self-fulfilling prophecy that the final impact will be minimal. The future? By 1990 (or 2000 at the latest), there should be a number of new and outstanding examples of true computer-based/-derived/-oriented projects in education, rather than a mere continuation of the current computer-augmented, yet traditional approach.

85

THE OPEN UNIVERSITY

Frank Lovis
The Open University
Milton Keynes, England MK7 6AA
(0908) 653371

William Dorn
Mathematics Dept.
University of Denver
Denver, CO 80210
(303) 753-3529

ABSTRACT

In 1982 Britain's Open University will replace the two second-level computing courses which it has been running since 1973. There will be only one new course since we have come to realize that running two second-level courses is unnecessary and extravagant. The new course is expected to maintain the current enrollment of 2,700 students per year.

Production of the new course is well underway, and in this paper its content and presentation will be compared with those of its predecessors. The new course was designed after careful consideration of the views of both students and faculty members from which emerged the one urgent and clear demand that the course should relate closely to the commercial world of data processing.

The main themes of the course are practical computing, data structures, files and file processing, systems analysis and design, and the social impact of computing. Introductions to operating systems, data bases, and distributed computing are also included. The course components comprise written materials, computer programming activities, 16 television programs, three teaching audio-cassettes, and six broadcast radio programs.

The present paper will concentrate on:

(1) Why the DP bias was considered essential and how it shows up in the course.

(2) The content and presentation of the first four main themes listed above. In this connection the systems analysis and design package bears special mention, as it is being developed in collaboration with the U K.'s National Computing Centre and will involve the student in a practical project.

(3) The specification and future implementation of OUSBASIC, a computing language designed to enable the student to write and run structured programs in the unique--and awkward--distance teaching environment of The Open University.

The paper will be illustrated with two excerpts, of ten to fifteen minutes duratic each, from films of two of the television programs--The Introduction to Files and File Processing and Systems Analysis and Design.

Science and Engineering

DEMOGRAPHIC TECHNIQUES IN ECOLOGY: COMPUTER-ENHANCED LEARNING

A. John Gatz, Jr.
Department of Zoology
Ohio Wesleyan University
Delaware, Ohio 43015
(614) 369-4431

INTRODUCTION

Judging from the availability of CONDUIT programs on ecological topics (5 of 8 in biology in the Summer 1979 issue of *Pipeline*), there may well be more computer use in courses related to ecology than in courses in other areas of biology. For instance, valuable CONDUIT programs exist that aid students in learning about population growth using the logistic equation and related models, interspecific competition using the Lotka-Volterra equations, and energy flow through the trophic levels of various ecosystems. Additional areas in ecology are appropriate for computer-enhanced learning. In particular, the use of the computer to aid students in understanding life tables is described here. Such work not only helps students understand these complex tables, but also permits them to quickly and easily get a grasp of the consequences, in terms of both population growth rates and age distributions, of alternative reproductive strategies.

SIGNIFICANCE OF LIFE TABLES

In many ways, a survivorship and fertility life table represents the ultimate synthesis of data on the life history characteristics of a given population or species of organisms. Both age specific death rates (l_x) and age specific birth rates (m_x) must be known and transcribed into appropriate format. For survivorship data, this means knowing the proportion of individuals surviving at the start of each age interval; and for fertility data, this means recording the number of female offspring per female in each age interval. From these data, numerous descriptive functions can be readily calculated by formulae of varying degrees of complexity (see Mertz, 1970 or Krebs, 1978 for a fuller description of these quantities). These functions or quantities include the following.

(1) The gross reproductive rate, G.R.R., is a hypothetical quantity that indicates the multiplication rate per generation if females suffered no mortality prior to completing reproduction at the rates specified in the fertility tables.

$$\text{G.R.R.} = \sum_{x=0}^{\infty} m_x$$

(2) The net reproductive rate, R_0, is the actual multiplication rate per generation if the population follows both the mortality and fertility schedules in the life table.

$$R_o = \sum_{x=0}^{\infty} l_x m_x$$

(3) The mean length of a generation, G, is given by:

$$G = \frac{\sum l_x m_x x}{R_o} .$$

(4) The innate capacity for increase in the particular environmental conditions for which the table was written, r_m, is a factor that gives instantaneous growth rates. The calculation of r_m must be done by trial and error in the formula:

$$\sum_{x=0}^{\infty} e^{-r_m x} l_x m_x = 1.$$

(5) The finite rate of increase, λ, is the factor by which population size changes per age interval in the life table, e.g., per year if the life table is in years.

$$\lambda = e^{r_m}$$

(6) The stable age distribution is the proportion of organisms that would be in each age category if the population continued to grow indefinitely according to the schedules in the life table. The proportion of organisms in each age category x to x+1 is given by

$$C_x = \frac{\lambda^{-x} l_x}{\sum_{i=0}^{\infty} \lambda^{-i} l_i}$$

where i, like x, is a subscript indicating age.

While the data that are used to generate a life table in the first place are quite intuitively understandable, e.g., the number of female offspring produced by an average six year old female, the derived terms, especially r_m, are much less easily grasped. Because a thorough conception of r_m is desirable in concrete terms (such as numbers of offspring and numbers of seasons of reproduction) before a student starts working with logistic growth models, a thorough familiarity with all the various life table parameters is highly important.

RATIONALE FOR COMPUTER USE

The reasons for using the computer in life table analyses can be divided into arguments against hand calculation and arguments for the computer. Considering first the arguments against hand calculation, there is the unfortunate, but nonetheless real truth that many undergraduate students are terrified by the sorts of summation signs and exponents intrinsic to several of the demographic formulae. At best, such students may laboriously plug through their calculations and then finish with absolutely no confidence in the accuracy of their results. At worst, some of these students may not even be able to perform the calculations. Clearly, neither of these situations is conducive to students' learning about ecology. On the other hand, for undergraduates capable of manually performing the calculations required, such an exercise rapidly becomes a numbingly mechanical process that, too, is not conducive to their learning about ecology.

As for the arguments for use of the computer, three points can be made. First, for students ever more used to punching numbers into either calculators or computers, one additional application of the computer generates very little anxiety. Math phobias can stay submerged. Second, the fast and accurate results provided by the computer maintain both the interest and confidence of the student. Moreover, if the student wishes to test himself and his ability to manually arrive at the same answers, the computer output does provide such a check. Finally, because the tedious parts of the exercise are done by a tireless computer, the student is free to concentrate on interpreting the biological significance of his findings. This is precisely the concentration that most ecology professors presumably desire, and I have found) useful in the present instance.

DESCRIPTION AND APPLICATIONS OF LIFET

With the above points in mind and two years' experience teaching about life tables without the aid of a computer, I wrote a short program, LIFET, in BASIC for our PDP 11/70 at Ohio Wesleyan University. LIFET would be suitable for use on other similar RSTS/E time-sharing systems or on minicomputers, and a program listing is available on request. In brief, the program asks the student to enter the age specific survivorship and fertility data for the population in question and then proceeds with calculations of the first three of the six quantities defined above. The innate capacity for increase is calculated by trial and error with active participation by the student in the trials. When the student is satisfied with the value of r_m that he has attained, the program uses that value to calculate the finite rate of increase. Finally, the student is given the option of obtaining a stable age distribution or not. A sample program run is given in Figure 1.

There are at least three sorts of application of this program. First, and most simply, is merely giving the students some real data from a natural population (e.g., from Vinegar, 1975 or Medica and Turner, 1976). The students can organize the data into life table format for

```
RUN LIFET
DO YOU WANT AN EXPLANATION OF THIS PROGRAM (YES OR NO)? YES

THIS PROGRAM CALCULATES LIFE TABLES AND DEMOGRAPHIC PARAMETERS. THE ONLY
INFORMATION NEEDED TO MAKE THESE CALCULATIONS IS THE AGE-SPECIFIC SCHEDULE OF
DEATHS AND BIRTHS WHICH YOU SUPPLY AS l(x) and m(x) VALUES WHERE:
      l(x) = PROPORTION OF POPULATION ALIVE AT THE START OF AGE INTERVAL x; AND
      m(x) = AVERAGE NUMBER OF BIRTHS TO EACH FEMALE ALIVE AT AGE x

TO RUN THIS PROGRAM, MERELY ENTER THE MAXIMUM AGE TO WHICH FEMALES SURVIVE
(RECALL THAT LIFE TABLES ARE FOR FEMALES ONLY) AND THEN FOR EACH AGE CATEGORY
ENTER THE l(x) and m(x) VALUES SEPARATED BY A COMMA. THESE DATA CONSTITUTE
ALL THE INPUT NECESSARY TO MAKE DEMOGRAPHIC CALCULATIONS USING THE FORMULAE IN
KREBS (1978, pp 160-170). SEE THIS REFERENCE FOR DEFINITIONS AND EQUATIONS.
ONLY THE EXACT r CANNOT BE CALCULATED WITHOUT ADDITIONAL INPUT ON YOUR PART.
THIS IS BECAUSE THE EQUATION MUST BE SOLVED BY TRIAL AND ERROR, i.e., BY
SUBSTITUTING ONE VALUE FOR r IN THE EQUATION
           -rx
      SUM e    l(x) m(x) = 1
AND THEN SEEING HOW CLOSE THE SUM COMES TO ACTUALLY BEING EQUAL TO 1.
FOR YOUR FIRST TRIAL, USE THE VALUE FOR r APPROXIMATED BY Ro AND G. IF ALL
REPRODUCTION TOOK PLACE IN THE SAME YEAR, THIS WILL ALSO BE THE EXACT r. IF
THE SUM YOU GET IS GREATER THAN 1, INCREASE YOUR ESTIMATE OF r. IF THE SUM IS
LESS THAN 1, DECREASE YOUR ESTIMATE OF r. INITIALLY MAKE FAIRLY LARGE CHANGES
IN r AND ONLY LATER MAKE MORE SUBTLE CHANGES AS YOU ADJUST THE SUM TO
1 +/-0.005. ONCE YOU ARE SATISFIED WITH YOUR VALUE FOR r, STOP ADJUSTING IT
AND THE PROGRAM WILL CONTINUE TO MAKE OTHER CALCULATIONS USING THIS FINAL
VALUE OF r ON WHICH YOU HAVE DECIDED.

WHAT IS THE LAST AGE AT WHICH SOME FEMALES ARE ALIVE? 3
FOR X = 0 WHAT ARE THE OBSERVED VALUES FOR
l(x),m(x)? 1.0,0
FOR X = 1 WHAT ARE THE OBSERVED VALUES FOR
l(x),m(x)? .9,2
FOR X = 2 WHAT ARE THE OBSERVED VALUES FOR
l(x),m(x)? .7,1
FOR X = 3 WHAT ARE THE OBSERVED VALUES FOR
l(x),m(x)? .5,1

THE GROSS REPRODUCTIVE RATE, G.R.R. = 4
THE NET REPRODUCTIVE RATE, Ro= 3
THE GENERATION TIME, G= 1.56667
AS APPROXIMATED BY Ro and G, r=.701242

THE PROGRAM WILL NOW CALCULATE AN EXACT r by TRIAL AND ERROR.
WHAT IS THE VALUE FOR r THAT YOU WISH TO TRY? .7
SOLVING THE EQUATION WITH r= .7 GIVES AN ANSWER OF 1.1277
DO YOU WISH TO TRY AGAIN WITH ANOTHER VALUE FOR r(YES OR NO)? YES
WHAT IS THE VALUE FOR r THAT YOU WISH TO TRY? .8
SOLVING THE EQUATION WITH r= 8 GIVES AN ANSWER OF .995479
DO YOU WISH TO TRY AGAIN WITH ANOTHER VALUE FOR r(YES OR NO)? YES
WHAT IS THE VALUE FOR r THAT YOU WISH TO TRY? .79
SOLVING THE EQUATION WITH r= .79 GIVES AN ANSWER OF 1.00784
```

Figure 1. Sample output from LIFET

Figure 1 continued

```
DO YOU WISH TO TRY AGAIN WITH ANOTHER VALUE FOR r(YES OR NO)? YES
WHAT IS THE VALUE FOR r THAT YOU WISH TO TRY? .795
SOLVING THE EQUATION WITH r= .795 GIVES AN ANSWER OF 1.00164
DO YOU WISH TO TRY AGAIN WITH ANOTHER VALUE FOR r(YES OR NO)? NO

THE FINITE RATE OF INCREASE, LAMBDA= 2.21444
THE EXACT INSTANTANEOUS RATE OF INCREASE FOR THE POPULATION, r= .795

DO YOU WISH TO CALCULATE THE STABLE AGE DISTRIBUTION FOR THIS POPULATION? YES

GIVEN A STABLE AGE DISTRIBUTION, THE PROPORTION OF ORGANISMS
IN EACH AGE CATEGORY, C(X), WOULD BE:
C( 0 ) =  .626875
C( 1 ) =  .254776
C( 2 ) =  .089485
C( 3 ) =  .288641E-1

 x          l(x)       m(x)        l(x)m(x)
 0           1          0           0
 1           .9         2           1.8
 2           .7         1           .7
 3           .5         1           .5
G= 1.56667             GRR= 4      Ro= 3

r= .795     lambda= 2.21444

DO YOU WANT TO CALCULATE ANOTHER LIFE TABLE (YES OR NO)? NO
```

themselves if desired, or the data can be presented as a table from the start. Either way, the students have the excitement of finding out for themselves whether various populations in nature are growing or shrinking and at what rates. If population growth is occurring and data are also available on numbers of individuals alive in each age class, the students can also investigate whether or not such population growth has been going on at the given rates for a number of years. If it has, the observed proportions of individuals in each of the age classes should be the same as the proportions given by the LIFET calculations of the stable age distribution. Students seem to enjoy checking this assumption of life table methodology.

A second application is the use of data not just from one population of a species, but from multiple populations of a single species. The work by Tinkle and Ballinger (1972) on intraspecific comparative demography of a lizard is especially appropriate. By analyzing the age specific death rates and fertility rates using LIFET, students can enhance their understanding of the population consequences for real animals of variations in life history strategies. These particular data, in fact, indicate that some of the natural populations are just holding their own and others are decreasing in size. With analyses completed, students can either write about or discuss hypothetical alterations in reproductive strategy that could improve the lot of the marginally surviving populations.

A third and final example of an application for this program is to give students free reign to devise an optimal reproductive strategy in each of several habitats for a totally hypothetical organism. A few guidelines need to be laid down at the outset, and then students can be set free to use their creativity. For example, one needs to specify how many kilocalories total a female has to produce eggs in her lifetime, the range in sizes (caloric contents) of eggs to be permitted, and the harshness of the various habitats in terms of probabilities of the survival of offspring to age 1 from each possible egg size in each habitat. Survivorship thereafter can either be specified or left to the students' own devices. The generation of an optimal strategy, or even possible strategies, for each environment then requires that students generate a number of life tables. Successful completion of such an exercise, in my experience, results in an excellent appreciation of the merits and drawbacks of iteroparity and semelparity in various situations.

SUMMARY AND CONCLUSIONS

For the past two years, I have used an interactive computer program, LIFET, as a means of enhancing my instruction of demographic techniques in an undergraduate course on animal ecology. I have found that I have been able to cover more examples and nuances of life history studies since I initiated the program than I was able to before. By obviating the time-consuming busy work of life table calculations, I can realistically expect my students to accomplish far more sophisticated and extended problems than were possible before I wrote LIFET. In my opinion, students who have had the benefit of using this program have gained a far greater understanding of life tables and demography than did my earlier students. Certainly, they have had more opportunities to do so in a challenging way. What's more, they enjoy it. This last point, in and of itself, can be a strong recommendation.

REFERENCES

Krebs, C. J. 1978. Ecology: The Experimental Analysis of Distribution and Abundance. 2nd ed. New York: Harper & Row. 678 p.

Medica, P. A., and F. B. Turner. 1976. "Reproduction by *Uta stansburiana* (Reptilia, Lacertilia, Iguanidae) in Southern Nevada." J. Herpetol. 10: 123-128.

Mertz, D. B. 1970. Notes on methods used in life-history studies. In: Readings in Ecology and Ecological Genetics, J. H. Connell, D. B. Mertz, and W. W. Murdoch (eds). New York: Harper & Row. pp. 4-17.

Tinkle, D. W., and R. E. Ballinger. 1972. "*Sceloporus undulatus*: A Study of the Intraspecific Comparative Demography of a Lizard." Ecology 53:570-584.

Vinegar, M. B. 1975. "Demography of the Striped Plateau Lizard, *Sceloporus virgatus*." Ecology 56:172-182.

MICROCOMPUTERS AS LABORATORY INSTRUMENTS: TWO APPLICATIONS IN NEUROBIOLOGY.

Richard F. Olivo
Department of Biological Sciences
Smith College
Northampton, Massachusetts 01063

The low cost of microcomputers has brought them into the same price range as ordinary laboratory instruments, and their ability to collect and store data makes them an extremely welcome addition to a laboratory. At Smith College, we have developed two applications for using microcomputers in neurobiology. One application is suited to advanced students and researchers and involves the use of a microcomputer and a digitizing tablet to measure photographic data; the other is suitable for routine use in a neurophysiology course and involves an analog/digital interface for collecting and displaying transient data. In describing these two applications, I shall emphasize a number of aspects that I believe are of general interest in introducing microcomputers into undergraduate laboratories.

HARDWARE: THE AIM 65

I chose Rockwell's AIM-65 microcomputer for laboratory use. Like its smaller cousins, the KIM and SYM, AIM-65 is a relatively inexpensive single-board computer that is based on the 6502 microprocessor. The AIM includes an input/output interface with two 8-bit parallel ports, which we use to connect the computer to laboratory instruments, plus two timers and an interrupt register. The AIM also has a full keyboard, a 20-character alphanumeric display, and a thermal printer. I consider it an advantage that the AIM does not use a video monitor, since video would make the system less compact and more expensive; the AIM's one-line display is adequate for prompts and data, and the output ports provide a means (with a digital-to-analog converter) of plotting data at high resolution on an oscilloscope or chart recorder. The AIM's printer further provides each group of students with inexpensive hard copy.

The AIM is also relatively convenient to program. It has an extensive (8K) monitor that incorporates a versatile editor, a disassembler, and a pseudo-assembler that permits writing programs in mnemonics rather than op codes. A full symbolic assembler, which I used for writing our programs, is optional, as is an 8K BASIC (both of these are supplied as read-only memories that plug into designated sockets on the AIM board). The assembler and BASIC sockets will also accept 2716 erasable, programmable read-only memories (EPROMs). Once a program is debugged, it can be loaded into an EPROM to be installed in the assembler socket, permitting a student to run a program by typing "N," the monitor call to the assembler. The student does not need to be an accomplished computer user; the programs are highly interactive and provide abundant prompts for entry of data and for menu choices.

In addition to the AIM, its enclosure, and a power supply, two other pieces of hardware are necessary. The AIM's 4K bytes of on-board memory are not sufficient for extensive digitization, so that a supplementary memory board is desirable. I chose the Memory Plus, an 8K board from The Computerist (P.O. Box 3, S. Chelmsford, MA 01824), which has the additional feature of an EPROM- programming circuit. Other manufacturers also make AIM/KIM-compatible 8K and 16K memory boards. The other major piece of hardware required is an analog/digital interface. Commercial A/D boards that can interface to the AIM's input/output ports are available, but the ones that I am aware of are too slow for digitization at the rates needed (about 10 kHz) in a neurobiology lab. Consequently, we built our own analog interface, which I shall describe later. At present we have two AIMs at work. One is a prototype system with 12K of memory and a single analog input/output channel; its cost is about $1000, and we are seeking funds to buy and build six more such systems for routine class use. The other AIM has 4K of memory and is devoted exclusively to taking data from a HiPad digitizing tablet (Houston Instruments). That system's cost, including the digitizing tablet, is about $1400. For comparison, these prices are of the same order of magnitude as a laboratory oscilloscope or a chart recorder.

APPLICATION 1: INTERFACING TO A DIGITIZING TABLET

I shall first describe our system for collecting data using a HiPad digitizing tablet. The program has been through several stages of design and is now in heavy regular use. It illustrates several aspects of interactive laboratory computing that I believe are of general importance.

Neurophysiologists typically measure the electrical changes in nerve or muscle cells in response to stimuli. The stimuli and responses are displayed on an oscilloscope screen and are usually photographed on 35-mm film by an automatic camera. The photographs serve as primary data for subsequent analysis of the traces and for illustrations for publications. Analysis requires measuring the amplitudes and time courses of the events, which is usually done by projecting the film in an enlarger onto graph paper and (before microcomputers) tediously counting squares. A digitizing tablet under the graph paper is a huge improvement; it takes data automatically, speeds up the analysis by a factor of at least ten, and is intrinsically more accurate than hand analysis. For those who have never seen a small digitizing tablet, it consists of a flat surface (ours is 11 inches square) in which wires are embedded, plus a cross-hair cursor that is placed on the surface and returns the X,Y coordinates of its position. In using a digitizing tablet, two problems must be solved: interfacing the microcomputer to the tablet, and creating an efficient program for recording and analyzing data. I shall describe the interfacing aspects first.

The electronics in the HiPad tablet output data in several formats, any of which may be suppressed by jumpers at the output connector. Different strobe lines are available for each of the formats, so that a microcomputer can be made to attend to one format and to ignore the others. I chose to take data in binary-coded decimal (BCD) form rather than in straight binary (hexadecimal), the alternative parallel format. The main reason for this choice is that the final output had to be in decimal form since hexadecimal data are incomprehensible to most biologists. I believe that an important principle exists here: a student or researcher should never be asked to read or generate hexadecimal numbers. Since the 6502 microprocessor can be set to do BCD arithmetic, it made sense to start, compute, and end in BCD rather than to convert to or from hex.

BCD DATA FORMAT

CODE DESCRIPTION	BIT POSITION								BYTE NO.
	MSB							LSB	
Control Word	1	1	1	1	X	X	X	X	1st BYTE
X Axis	0	0	Sign		MSD				2nd BYTE
X Axis	2nd SD				3rd SD				3rd BYTE
X Axis	4th SD				LSD				4th BYTE
Y Axis	0	0	Sign		MSD				5th BYTE
Y Axis	2nd SD				3rd SD				6th BYTE
Y Axis	4th SD				LSD				7th BYTE

DATA TIMING

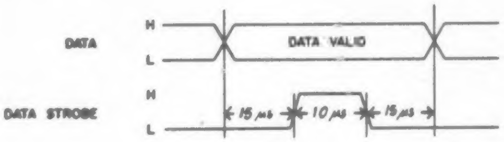


FIGURE 1.

In its BCD format, the HiPad tablet outputs seven bytes of data for each digitized point. The contents of these seven bytes are shown in Fig. 1, which also shows the timing of the data. Note that byte 1 is a control word, bytes 2, 3, and 4 contain the X-coordinate, and bytes 5, 6, and 7 contain the Y-coordinate. The timing diagram shows that the BCD strobe line goes high as each byte is output. Thus, a subroutine to read data into the microcomputer must test for the presence of the strobe pulse, then read the current byte, store it, and repeat these operations until seven bytes have been read. I connected the BCD strobe to control line CA1, and the eight data lines to the AIM's Port A. In the 6502 system, input/output ports and associated registers are read from and written to as memory locations, so that operations on the strobe pulse and data resemble reading data from memory.

The full subroutine to take a data point is shown in Figure 2. When the subroutine is called, the microprocessor enters a three-instruction loop in which it remains until the strobe pulse occurs. In this loop, it checks the interrupt flag register (IFR, part of the 6522 input/output chip) to see if line CA1 is high. It then reads Port A, checks for the expected control word, and if it is found, proceeds to read and store the remaining six bytes of data. Once the six X,Y bytes have been stored, they are repacked to place the sign of each coordinate in a separate byte for more efficient handling. In the overall program, the GETPT subroutine is followed by another subroutine that tests whether the point just taken lies in a menu area of the digitizing tablet; if so, the program jumps to the appropriate section; otherwise the point is treated as a legitimate data point.

```
--------------------
GET POINT FR. HIPAD

==D5D1 GETPT
851CC9 LDA IFR
0D90   AND #$02
0FA9   BEQ GETPT
15D024 LDA PORTA
A90D   AND #$F0
1865   CMP #$F0
1C85   BNE GETPT
==D5E1
1CC9   LDX #00
==D5E3 STROB1
10B00D LDA IFR
A088   AND #$02
B966   BEQ STROB1
DE1865 LDA PORTA
1020   STA XYDATA,X
D8     INX
D810   CPX #$06
20A5   BNE STROB1
==D5F4
;REPACK DATA
1DF0   LDY 04
;SHIFT 4 BITS
==D5F6 SHFDAT
17A5   LSR XYDATA
30C9   ROR XYDATA+1
02F0   ROR XYDATA+2
05A9   LSR XYDATA+3
104C   ROR XYDATA+4
76D6   ROR XYDATA+5
A9     DEY
0105   BNE SHFDAT
30     RTS
```

FIGURE 2.

```
EXPT DATE: 11/28/79
--------------------
--------------------

CURRENT?(Y,N):    Y
V-HALF?(Y,N):     Y
SCALE,MS/DIV:  0200
      NA/DIV:  1000
      MV/DIV:  0020

--------------------
FRAME 0030    ID# A1
CLOCK 18:30 11/28/79
--------------------
IBASE +0020 NA
      +0000 MS
VBASE +0034 MV
      +0004 MS

IPEAK +3860 NA
      +0156 MS
VPEAK +0094 MV
      +0228 MS
VHALF +0047.20 MV
      +0664 MS


--------------------
FRAME 0031    ID# A1
CLOCK 21:30 11/28/79
--------------------
IBASE +0000 NA
      +0004 MS
VBASE +0031.60 MV
      +0004 MS

IPEAK +3140 NA
      +0156 MS
VPEAK +0095.60 MV
      +0228 MS
VHALF +0048 MV
      +0640 MS
```

FIGURE 3.

In addition to solving these aspects of interfacing, it also is necessary to have an efficient program for data collection. My assembly-language program generates almost 2K bytes of object code, a substantial amount. On initial entry, various constants and tables are set equal to zero, after which the date of the experiment and several scale factors are requested by prompts to the user. The main part of the program then begins; in this section, a series of points is taken for each frame and stored in tables in memory. Figure 3 shows the print-out made during the early part of running the program, and Figure 4 shows part of the tables that are printed after all frames have been measured.

Several aspects of this program are of general interest. First, every measurement that the user must make is preceded by an explicit prompt on the AIM's display. Prompts make the program easy to use and are always good practice in interactive computing. Second, the program can be re-entered without going through the initialization routine; existing data are thereby retained in memory unless they are deliberately erased. Easy re-entry makes the program more forgiving of errors, such as accidental escape to the monitor, and it also makes it possible to print tables or re-enter scale factors at intermediate stages in the measurements. Re-entry is accomplished by two means. The first was mentioned above: the left edge of the digitizing tablet is reserved for a menu, and each point taken is tested to see if it falls in the menu area. The menu is available whenever the program is expecting a data point, which in practice is most of the time. The other method of re-entry uses the three user-defined keys on the AIM's keyboard. During the initialization routine, these keys are programmed to initiate jumps to certain sections of

```
--------------------    --------------------
EXPT DATE: 11/28/79     EXPT DATE. 11/28/79


ID=A1                   ID=A1
FR# VBMV  IPNA IPMS     FR# VPMV  V/2MS TIME

030 34.00 3860 0156     030 094.00 0664 1830
031 31.60 3140 0156     031 095.60 0640 2130
032 31.60 2900 0156     032 007.60 0504 2230


ID=A2                   ID=A2
FR# VBMV  IPNA IPMS     FR# VPMV  V/2MS TIME

034 30.40 3460 0156     034 007.60 0660 2400
035 30.00 3560 0156     035 003.60 0732 2500


ID=A3                   ID=A3
FR# VBMV  IPNA IPMS     FR# VPMV  V/2MS TIME

037 30.00 2800 0156     037 090.00 1492 2630
038 30.40 3000 0156     038 092.40 1264 2730


ID=B1                   ID=B1
FR# VBMV  IPNA IPMS     FR# VPMV  V/2MS TIME

045 32.00 2880 0160     045 090.40 0708 3300
046 28.40 3960 0160     046 093.20 0472 3400
047 32.00 3660 0156     047 090.00 0528 3500
```

FIGURE 4.

the program. Thus, the user can redirect the execution of the program fairly conveniently and without losing data already entered.

Another aspect of general interest is that for each frame measured, the program stores data in a temporary buffer until the last measurement for that frame is taken. The data are then transferred automatically to the tables in memory. The point of the temporary storage is to allow the user to escape and remeasure a frame without contaminating the final tables if an error in measurement is suspected. Anyone who has used a program that does not permit correction at the time of entry knows the frustration that accompanies being forced to continue in a process one knows to be incorrect. In this case, by permitting a penalty-free escape in the middle of a frame, a compromise is achieved between ease of programming and no correction routine at all; relatively few measurements are made in one frame, so that repeating a frame is not onerous, and the final tables remain accurate.

In summary, this application of microcomputers to data collection illustrates several aspects that I believe are characteristic of good interactive programs. The program is permanently in an EPROM and can thereby be called by a single keystroke, without requiring that the user know how to run a tape- or disk-loading routine; also, it cannot be accidentally written over. The program provides abundant prompts. Data input and output are always in decimal format. The program can be re-entered and redirected without erasing existing data, making it more convenient and more forgiving of errors. Errors of which the user is aware can be corrected at the time of entry. Each of these features is, I think, important in bringing microcomputers into the laboratory. The philosophy behind this approach is to make the computer friendly to a user who is not expert in programming. As a colleague expressed it, "You don't expect students to build an oscilloscope in order to use one, so why expect them to program the microcomputers that they use?"

APPLICATION 2:
ANALOG-TO-DIGITAL CONVERSION

Our second application will eventually be used by more students than the first one, but the program is still in an early stage, so I will describe it only briefly. Students in neurobiology laboratories typically record highly-amplified, transient electrical signals from the nerves of animals such as frogs or crayfish. The signals are observed on oscilloscopes, and a recurrent difficulty in the past has been that the transient signals are difficult for students to measure or even observe. Storage oscilloscopes are helpful, but they are expensive and they do not provide hard copy. Routine photography is slow, awkward, and expensive, particularly if Polaroid film is used. Microcomputers, on the other hand, offer the possibility of digitizing signals and playing them back for analysis. Playback can be repetitive and fast for observation on an oscilloscope, or it can be one-shot and slow for writing out on a chart recorder. In addition, in many experiments signals are recorded from single nerve cells rather than from groups of tens or hundreds of nerve cells. In such cases, the information of interest is the time interval between individual electrical impulses (action potentials) in a nerve cell. Histograms of inter-pulse intervals and post-stimulus firing rates are routinely computed in research laboratories; we can now make such techniques available to undergraduates as well. Finally, some neurophysiological experiments (such as recording evoked potentials from the scalp) require signal averaging because the signals are much smaller than the background electrical noise. Once again, microcomputers can make signal-averaging techniques available to undergraduates.

Digitizing transient signals and signal-averaging both require analog-to-digital conversion, but presently available A-to-D boards that interface to a parallel port are too slow for use in a neurobiology lab. An action potential can be as brief as 1 msec, and if its shape is to be preserved in digitization, at least 10 samples should be taken. This requires a sampling interval of 100 usec. Analog Devices (P.O. Box 280, Norwood, MA 02062) manufactures an integrated-circuit, 8-bit A-to-D device that can make one conversion every 25 usec and that costs less than $25. I have used their device in the analog converter board whose block diagram is shown in Figure 5. The converter requires two control lines, one to start conversion and one to signal that the data are ready, plus eight data lines. These lines are connected to Port A and to CA1 and CA2 of the AIM. Port B is connected to a digital-to-analog converter, for playback of digitized data. The Port B control lines, which are not otherwise needed, are used for input and output of trigger pulses.

My interactive program for A/D conversion, like the HiPad program, will reside in an EPROM and will make the microcomputer seem like a smart instrument to the user. The major input routine provides continuous digitization with scrolling display, so that the oscilloscope screen resembles a moving chart. Receipt of a trigger pulse causes digitization to cease after a preset interval, followed by continuous replay of the last three pages of data (three pages are about the limit for a flicker-free display). The user can also choose to output the digitized data slowly, for playback to a chart recorder. Chart recorders have frequency passbands from DC to 100 Hz, at most, while the original signal has important components up to about 5 kHz; thus an expansion of the time-scale by a factor of 100 is necessary to write the data accurately on a chart recorder. The time-scale can be expanded easily by inserting a timing loop in the playback program.

Digitization at 10,000 samples per second would appear to require huge amounts of memory, but a comparison with the oscilloscope sweep speeds that are used by neurobiologists to display data shows that the memory requirements are not excessive. A single sweep typically displays from 10 msec to 0.5 sec of data, so that 8K bytes of memory would hold two to eighty sweeps. In many cases, when longer sweeps of data are of interest, high rates of digitization are not needed, and the effective capacity of the memory can be extended.

Thus, in summary, the relatively low cost of microcomputers opens many new possibilities for using them in the laboratory. The programming effort that is required can be extensive, and some of the interface hardware may have to be constructed, but the results give students enormously greater capabilities for analyzing data. It is important that undergraduates in science receive experience with such techniques, for they will work in a world in which computerized collection and analysis of data will be routine.

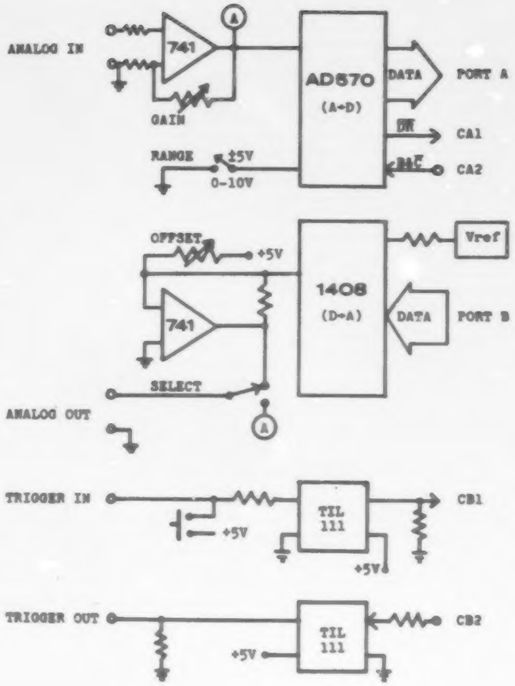


FIGURE 5.

96

CLASSICAL MECHANICS WITH COMPUTER ASSISTANCE

A. Douglas Davis

Department of Physics
Eastern Illinois University
Charleston, IL 61920

This paper describes the use of a computer and elementary numerical analysis in the teaching of a course in classical mechanics. This is a traditional, rigorous, calculus-based mechanics course. The use of the computer allows students to solve problems somewhat before the analytical solution is developed. Such prior solution allows them to anticipate the analytical solution and greatly aids in their understanding. Computer-generated solutions also allow the investigation of interesting problems whose analytical solution would otherwise be beyond the scope of this course.

INTRODUCTION

This paper describes the use of a computer and elementary numerical analysis in the teaching of a course in classical mechanics. This is not a computer-based mechanics course. Rather it is a traditional, rigorous course in classical mechanics. Computer-generated solutions are used as simply one more tool to teach the real physics of the situation.

The first week (three 50-minute lectures) is devoted to teaching "conversational BASIC"-- just enough BASIC that even students with no prior exposure to computers can go to a terminal and write the simple programs we shall use. Thus, students can immediately write, run and change their own BASIC programs. The TAB function in BASIC allows even neophytes to obtain graphic results very quickly. In addition, the results of some instructor-written computer programs are used. Both student-written and instructor-written programs give students another tool-- another point of view, another handle -- to use in developing physical intuition and a solid understanding of what's going on in a given situation.

Computers are used in the following areas:
1. Introduction to variable forces.
2. Introduction to integral calculus.
3. Investigation of harmonic motion.
4. Alternate approach.
5. Central force orbits.

INTRODUCTION TO VARIABLE FORCES

In any good, solid introductory physics course students will have solved essentially all of the basic problems involving motion caused by a constant force. Variable forces require the use of calculus for a solution and are usually not covered in detail in an introductory physics course - even if it is nominally calculus-based.

Harmonic motion is the motion of a body under the influence of a linear restoring force and can be written as $F = -KX$ where F is the force, X is the position, K is the spring constant which determines the strength of the force, and the negative sign indicates that the force always acts to move the body back to the origin. A classic example is a body of mass M attached to a spring with spring constant K. But harmonic oscillators have more wide spread use than that. A thorough understanding of harmonic oscillators is useful, even necessary, for understanding such diverse things as automobile suspension systems, radio receivers, and ultra-violet absorption by the atmosphere.

A major foundation of classical mechanics states that a force acting on a body will cause it to accelerate. This acceleartion is directly proportional to the force and inversely proportional to the mass and can be written in the form of a = F/m or, as is more usually done, in the form of F = ma. Beginning with this in the second week of class students write a simple iterative program to solve for the motion of a harmonic oscillator. The essentials of the program are:

```
100 LET F = - K * X
110 LET A = F/M
120 LET V = V + A * D
130 LET X = X + V * D
140 LET T = T + D
150 PRINT T, X, V
160 GO TO 100
```

where F is the force; K, the spring constant; A, the acceleration; V, the velocity; X, the position; T, the time; and D, the time increment ΔT. Additional details of the program allow D to be small for greater accuracy yet have only a manageable amount of data to be printed out. A more sophisticated numerical analysis routine, like the Runga-Kutta method, could be employed (1). But since the object of all this is to understand the physics of the motion rather than extreme numerical accuracy, the simpler method seems preferable. While classroom discussions are usually limited to writing a program in BASIC, this procedure is readily adaptable to a hand-held programmable calculator (2).

The TAB function in BASIC allows a student to get graphic output readily by changing the PRINT statement to:

```
150 PRINT TAB (40 + 30 * X); "*"
```

which centers the output on column 40 when X = 0 and has a scaling factor of 30 to PRINT an asterisk in column 70 or column 10 if X has a value of +1.0 or -1.0. If considerably larger or smaller amplitudes are expected the scaling factor is changed accordingly.

Such graphic output allows the students to see the motion -- and investigate its dependence upon various parameters -- in some detail before we begin the rigorous analytical solution of the same problem. Knowing more about the behavior of a system makes finding a mathematical solution all the easier and more meaningful once it is obtained by more traditional means.

INTRODUCTION TO INTEGRAL CALCULUS

The fact that an integral represents the area under a curve is of vital importance in physics. Yet students sometimes complete the average course in integral calculus knowing an integral as simply an operation, the antidifferentiation of a function. To stress the idea of an integral as the area under a curve or as an infinite sum, a homework problem is assigned that asks for the sum of the areas of small rectangles bounded by the quadrant of a circle. The quadrant of a circle can be broken into five, ten, perhaps 20 or even 100 small rectangles whose individual areas can be calculated by hand. As more and more smaller and smaller rectangles are used, their total area comes closer and closer to the actual area of a quadrant of a circle, $\pi r^2/4$ or $0.7854r^2$. An integral, since it is a sum of an infinite number of infinitesimally small rectangles gives *exactly* that result. To make this point unmistakably clear, the students are asked to continue by breaking this quadrant of a circle into 100, then 1000, and finally 10,000 tiny rectangles. This problem would be unrealistic and futile to attempt by hand, but is an easy and interesting problem for the computer.

INVESTIGATION OF HARMONIC MOTION

F = -KX describes a *simple* harmonic oscillator. Addition of a frictional damping force turns this into a much more realistic *damped* harmonic oscillator. Such a damping force might represent mass and spring wiggling under water, or in cold molasses, or the resistance in a radio or the shock absorbers on a car. Its inclusion drastically changes the technique and approach necessary for an analytical solution. But long before the students are concerned with the details of an analytical solution, they have amply investigated the behavior or motion of this damped harmonic oscillator. The only change to the earlier computer program is to redefine the force, including the damping force.

```
100 LET F = -K * X - C * V
```

With this change, the students can now investigate the motion for various initial conditions and various values of the damping coefficient C. Underdamping, critical damping, and overdamping become terms of real significance describing certain particular and characteristic motions of the oscillator.

Resonance phenomenon or the behavior of a *driven* or *forced* harmonic oscillator has application throughout technology. Tuning of a radio is but one example. Long before the students hear the ominous phrase "inhomogeneous second-order differential equation," they will have learned much about the characteristics of solutions to just that. Again, one simple change to the initial computer program is all that is required. The equation defining the force now becomes

```
100 LET F = -K*X-C*V+E*SIN (W*T)
```

where E is the strength of an external driving force which varies sinusoidally with angular frequency w. Resonance can clearly be seen and investigated by changing various parameters and observing the effect those changes have on the output.

ALTERNATE APPROACH

Some very interesting real-world problems are difficult to solve analytically. Others can be solved analytically without too much difficulty, but understanding the full meaning of the solution may not be entirely clear to students. For both situations a computer-generated solution is a very useful alternate approach.

It is an easy matter to discuss the trajectory of an object thrown and then acted upon by Earth's uniform gravitational field -- as long as friction through the air is neglected. That's a reasonable approximation for many situations. But it is also a rather interesting problem to include air resistance and then investigate the trajectory. Air resistance is not negligible if a student throws a crumpled wad of paper at a waste can or if a naval cruiser fires a shell at a target ten kilometers away.

It is an easy matter to return to the original computer program and modify it to handle a ballistic trajectory with linear air resistance:

```
100 LET F1 = -C * V1
105 LET F2 = -C * V2 - M * G
110 LET A1 = F1/M
115 LET A2 = F2/M
120 LET V1 = V1 + A1 * D
125 LET V2 = V2 + A2 * D
130 LET X = X + V1 * D
135 LET Y = Y + V2 * D
140 LET T = T + D
150 PRINT T, X, Y
160 GO TO 100
```

C is the air drag coefficient, G is the acceleration due to gravity, the quantities with a 1 suffix refer to horizontal or x-components, and quantities with a 2 suffix refer to vertical or y-components. This program yields a data table of horizontal and vertical positions which students are asked to plot in order to see the trajectory. Initial conditions of muzzle velocity and firing angle are varied; the air drag coefficient is also varied.

CENTRAL FORCE ORBITS

Gravitational forces, electrostatic forces, and forces on an isotropic harmonic oscillator are all examples of central forces, forces which depend only upon an object's distance from a certain origin. Central forces occur throughout nature. Any course in classical mechanics will spend a considerable amount of time and effort investigating central forces in general and the inverse-square law of Newton's Law of Universal Gravitation in particular. The orbits of interplanetary space probes, comets and planets all are both important and interesting.

Again, the computer can enhance this aspect of classical mechanics. For example, after the analytical solutions are derived and discussed, a desktop mini-computer with attached x-y plotter can be brought into class. As the class watches, orbits are drawn for various initial conditions. The conditions for a circular orbit become obvious, and the real meaning of escape velocity is made entirely clear.

This computer and plotter also allow the students a glimpse at how planets would move if the universe had been designed differently. Orbits for a force that has a radial dependence of

$$\frac{1}{r^3} \quad \text{or} \quad \frac{1}{r} \quad \text{or} \quad \frac{1}{r^{2.5}}$$

or whatever one likes can be handled just as readily as the $\frac{1}{r^2}$ of the real force. This always sparks students' interest and leaves them with an experience somewhat more tangible than just the discussion of an equation.

CONCLUSION

Classical mechanics forms a very important foundation in the education of physicists and engineers. And, almost invariably, it is considered difficult by students. Both student-written and instructor-written computer programs offer an additional tool to aid students in understanding mechanics. Students find the computer enjoyable (even those with little or no background who also find its use challenging). They report that its use greatly aids them in gaining a thorough understanding of the analytical solutions, for as they see how a system actually responds, the mathematics eventually derived to describe that motion becomes much more meaningful.

REFERENCES

(1) A F. Vierling, "Harmonic Motion", Computer-Based Physics: An Anthology, Commision on College Physics, 1969, Page 35.

(2) R. Eisberg, Applied Mathematical Physics with Programmable Pocket Calculators, McGraw-Hill, 1976.

COMPUTER AUGMENTED VIDEO EDUCATION IN ELECTRICAL ENGINEERING AT THE U. S. NAVAL ACADEMY

Michael W. Hagee
Tian S. Lim
Richard A. Pollak
United States Naval Academy
Annapolis, Maryland 21402
(301) 267-3492

ABSTRACT

This paper describes a computer-augmented video education (CAVE) project being undertaken in the Electrical Engineering Department at the United States Naval Academy. The project is designed to produce a series of modules involving computer graphics display and integrated computer-controlled television to help engineering students (non-electrical) as well as non-engineering students learn the essentials of electrical engineering. Each module contains a quick recap of the theory and basic sample problems, followed by an exercise to test students' proficiency.

I. INTRODUCTION

Although there are different areas in which a midshipman at the United States Naval Academy can major (from nuclear physics to English literature), all midshipmen regardless of major are required to take a two-semester survey course in electrical engineering during their second class (junior) year. Two different two-semester courses are offered. One is the core course program for non-engineering majors and the other is the engineering core course program for engineering majors. Both courses cover basically the same material at slightly different levels. Both courses are a mixture of theory and practical work and cover just about every major electrical engineering area.

This unique requirement for all students to successfully complete a two-semester college level electrical engineering course has presented some time-consuming problems. In the past, one-on-one extra instruction (EI) sessions have been the main tool in helping those non-technically oriented students.

EI sessions involving an instructor and one or two students have always been an important part of the learning process in the Department of Electrical Engineering and is in part a philosophy of teaching at the Naval Academy. The somewhat slower or weaker students find this type of aid essential in any course in which applying basic principles to solve problems is a primary objective and also used as a measure of achievement. These EI sessions were found, not surprisingly, to be remarkably similar. They begin with a quick recap of pertinent theory followed by the solution of a basic problem applying the principles just reviewed. As time permits, the problem solving is repeated with variations in the hope that the student builds both experience and confidence. In fact, the confidence of the students that they can succeed by applying the different principles to slightly different situations is of paramount importance in such a course. A typical EI sessions is also characterized by usually covering only one or two of the stated learning objectives of the program. Sessions covering the same material are repeated with different students. In this sense they are inefficient and tax the instructor not necessarily just for the time involved but also because even a skillful and patient instructor can find it difficult to maintain enthusiasm through constant repetition.

II. DESCRIPTION OF COMPUTER-SUPPORTED INSTRUCTION SYSTEM

The Academic Computing Center at the U. S. Naval Academy has developed a computer-supported instructional system (CSIS). It is intended for use as a drill/practice/tutorial tool to aid USNA instructors in computer presentation of their course material. The same learning material can also be used as a test. One feature allows the instructor to build an exercise in basic building blocks called frames. A frame may be in the form of a question, a comment, or a help sequence; or it may be a means of letting the student control branching. The frame-type designator determines how the computer is to analyze the student's response, if such a response is necessary. Another important feature is that frames may contain numeric and string variables and function expressions -- all under the control of the author/instructor. This feature allows the instructor to present the same functions (question types) to all students, but each student gets a different set of values for the variables and functions (questions) used in the frame. Furthermore, the instructor can control such things as the number of tries a student is allowed for answering a question correctly, whether the frame contains graphics, the number of times a frame is to be presented with different variables substituted, and allowable correct answers.

A team of eight instructors from the Department of Electrical Engineering is currently working on a project using CSIS. The current project will incorporate those phases of aid to students into packages or modules of material available at any time in the Academy's audio visual centers.

The project team is producing a series of modules of video and computer materials which could be used by students as a self-help, extra-instruction technique. Each module is keyed to one of the recurrent stumbling blocks encountered each year by non-engineering majors in their first course in electrical engineering. These modules are being designed to replace some of the regular extra-instruction tutored/taught by the faculty.

A study of past exams, student evaluations, and faculty questionnaires indicated that all students, from the superior to the weaker, experienced common stumbling blocks. Interestingly the problem areas were common to both the engineering and non-engineering students. These areas covered such topics as network reduction, Thevenin's Theorem, Norton's Theorem, voltage/current division, RC and RL transient analysis.

It is our conviction that the computer is most significantly exploited as an educational tool when it is used in conjunction with other activities. All module design work is approached with the idea of complementing class work with practical drill and practice exercises which reinforce the major learning objectives.

All extra-instruction modules are designed to give initially a brief (one or two frame) review of the objective in question. However, the heart of the modules is the drill and practice frames. These frames can select problems and circuits at random and through the use of predictable wrong answers (PWA) guide the student through the necessary steps to a solution. The instructor/programmer can incorporate video anywhere in the program, from using educational television for the initial introduction to having certain segments serve as help sequences for the PWAs. Since both the graphics terminal and the video cassette player are completely controlled by the computer, the student is relieved of all coordination tasks but is still an active part in a multi-media presentation. A control interface was designed at the Naval Academy to take signals from the author's program and physically control the video cassette player. There are commands such as "GO TO" and "PLAY." These commands transport the video tape under author/instructor or student control. The computer really acts as a coordinator. The student, not the computer, can control the instructional flow. This control results both from answers to presented problems and from input to periodic decision frames. Figure 1 shows an arrangement of the color television monitor, color video cassette player, and the computer graphics display terminal.

In what follows, we describe some of the computer-aided EI programs.

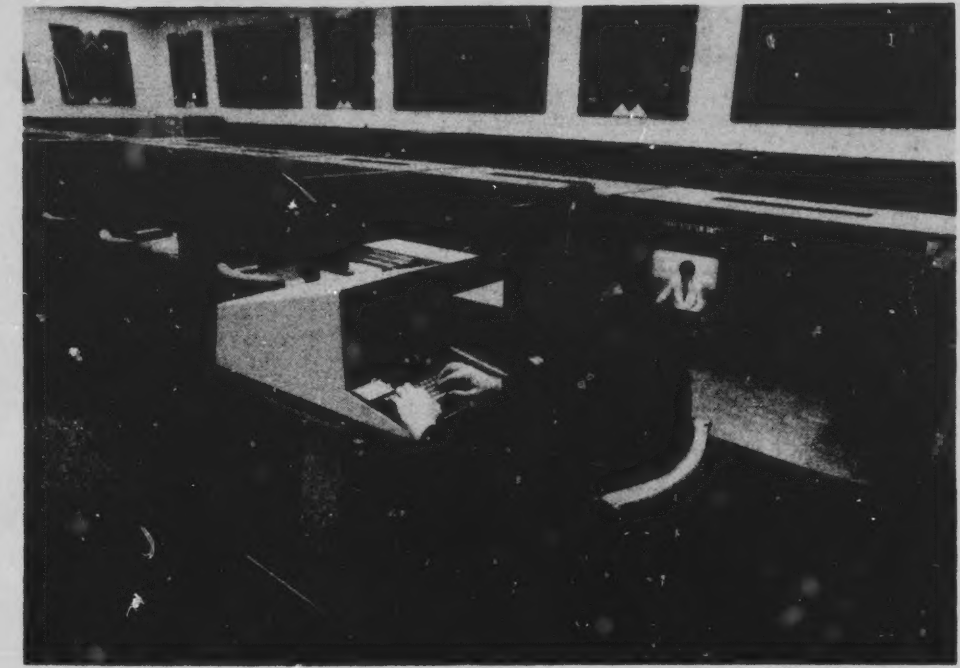
A. Voltage/Current Division

Figure 2 depicts a typical question frame block diagram. Figure 3 reflects a specific question frame in the voltage division module. As the student enters this particular frame, the program assigns random values from previously stored files to all circuit variables. The component across which the voltage drop is to be calculated is also randomly selected.

Based upon the values assigned and component selected, the program calculates the correct answer, six predictable wrong answers (PWA), and one unpredictable wrong answer. The student then enters his answer.

In Figure 4 the user has entered a value of 65, and the computer has responded with the unpredictable wrong answer message. In Figure 5 the student entered the correct magnitude but had the wrong polarity which caused one of the PWA messages to be displayed. Finally in Figure 6 the student has answered correctly and received an appropriate congratulatory response.

FIGURE 1



The PWAs are not limited to a one- or two-line message but can be expanded into several supplementary frames or a video presentation. Upon completion of these help sequences the user is usually given the same or similar questions to test his understanding.

After a successful answer, the program checks to see whether the student has received the required number of presentations. If not, a new circuit with different components and values would be randomly selected and presented.

If the number of presentation criteria is satisfied, the program determines whether the user has answered a predetermined number of questions successfully to continue to the next objective. If not the student is sent to another series of help sequences; otherwise he is directed to the next objective.

All variables, circuits, components, number of presentations, and number of attempts are under the author's control.

B. RL Transient Module

This program consists of 28 frames. As an example, suppose the student has reached Frame 8 and is presented with the circuit as shown in Figure 7. He is asked to answer the questions and the picture remains on the screen until he has provided an answer. If the answer is correct, he moves on to the next topic. If the answer is wrong the first time, he is given a second chance. If his second answer is still wrong, he is given three choices: (1) more review, (2) take the test again, or (3) stop. This information is given in Frame 13 as shown in the Appendix. All he has to do to make the choice is to type in REVIEW, CONTINUE, or STOP. If he chooses REVIEW, or CONTINUE, an appropriate picture will appear on the screen to provide the information he has asked for. If he chooses to STOP, he will be asked if he wishes to comment on the program. He can either type in "NO" (no comment) or make a comment and then sign off.

III. SUMMARY

Computer-augmented video education is a highly interactive process that is difficult to describe verbally. The best understanding can really only be achieved by sitting at a carousel/terminal and running the module. A good computer-based instruction system must satisfy several different communities: the educator-programmer, the instructor, and most importantly the student.

Student response to the first set of modules has been heartening, as a large number indicated that they found the modules to be helpful and enjoyable. Many students tried the programs after hearing that they were of help; on the night before each major exam approximately 25% of the class worked one or more modules. We feel it is important to develop programs that will be challenging and compelling and that will encourage students not only to try other topics, but also to recommend the programs to other students.

The twelve modules that were used covered material of about 1/4 of the course, and approximately 75% of the students in the course used one or more programs. It appears that the availability of additional topics would have resulted in a greater percentage of the students using the modules, since other topics were suggested by both users and non-users. The advisability of providing a complete set of supplementary programs is being studied at this time.

A total of six sections out of 45 were randomly selected to authenticate each module. Three of these sections were designated control groups while the remaining three were the test groups. Although data are still being analyzed, initial indications point to a high correlation between the test groups and the higher test averages. (The complete data analysis will be available by the time of the conference.)

There were two major problem areas pointed out by the students. The first, not surprisingly, was delays caused by slow computer response. The chaining process used in the display of graphics caused some fairly large delays especially when the number of users on the system was in excess of 100. This problem must be dealt with (it will be studied this summer), since these delays may have prevented some students from using the programs and could dampen enthusiasm for any interactive computer materials.

The second most often recorded comment was a pedagogical one. The students had a strong desire to see the school solution to a problem. This was true even though they may have answered the problem correctly. There was also a strong preference for seeing the specific problem missed worked in the help sequence rather than seeing a general review of the theory involved.

Through the use of computer graphics display terminals, well engineered and flexible software, and a system with a demonstrated high reliability, we believe we have developed a pedagogically sound and interesting set of modules that will complement the regular classroom instruction. The student can use the system to increase basic electrical engineering skills through drill and practice. The instructor can anticipate to spend less time on EI sessions and therefore can devote more time to students with serious difficulties.

This type of customized instruction offers the added benefit of diagnosing students' specific problems in given concept areas as well as measuring various teaching methods, since one can monitor the responses/comments of students to the given material.

REFERENCES
1. R. Pollak and J. Schwab, "USNA - Computer Supported Instruction System: Sophisticated, Generative and Easy to Author," Proceedings of the 16th Annual Meeting of Association for Educational Data Systems - Higher Education, 1978.

2. M. B. Sousa, "Computer Augmented Video Education," Educational Technology, February 1979, pp. 46-48.

FIGURE 2

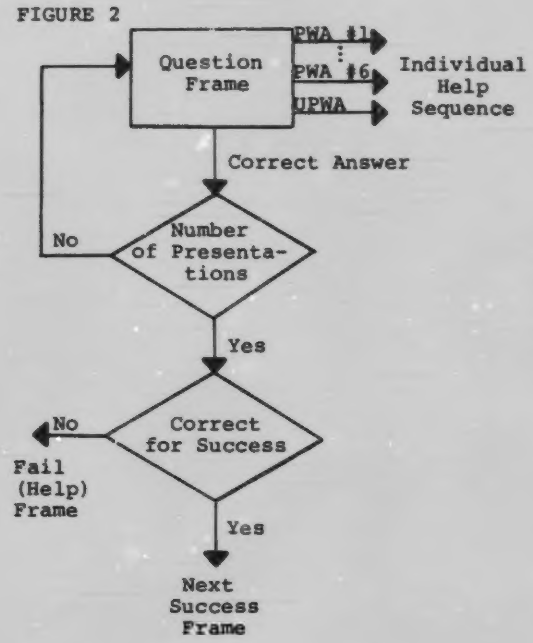


FIGURE 3

QUESTION #1 FRAME 51

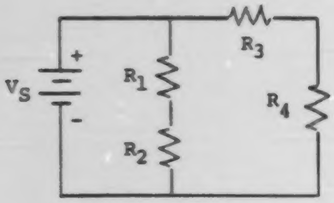
You will be given three chances to answer all questions correctly. If at any time you feel you would like to review the voltage divider technique concept, type "00" when a question is asked.

(The calculator mode may be entered by typing "$CS".) --Wait for the circuit.--

What is the value of the voltage drop across R2?

R1=1500 OHMS R2=670 OHMS R3=560 OHMS
R4=600 OHMS Vs=6.3 VOLTS

ANSWER?



CIRCUIT 7

FIGURE 4

QUESTION #1 FRAME 51

You will be given three chances to answer all questions correctly. If at any time you feel you would like to review the voltage divider technique concept, type "00" when a question is asked.

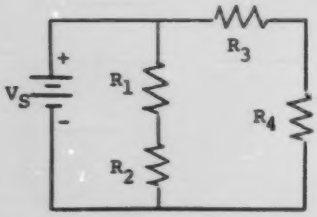
(The calculator mode may be entered by typing "$CA".) --Wait for the circuit.--

What is the value of the voltage drop across R2?

R1=1500 OHMS R2=670 OHMS R3=560 OHMS
R4=600 OHMS Vs=6.3 VOLTS

ANSWER? 65

I'm sorry but your answers don't agree.
I think your approach is incorrect.
Please TRY AGAIN or type 00 to review.



CIRCUIT 7

FIGURE 5

QUESTION #1 FRAME 51

You will be given three chances to answer all questions correctly. If at any time you feel you would like to review the voltage divider technique concept, type "00" when a question is asked.

(The calculator mode may be entered by typing "$CA".) --Wait for the circuit.--

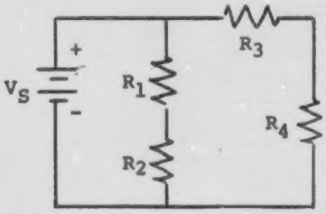
What is the value of the voltage drop across R2?

R1=1500 OHMS R2=670 OHMS R3=560 OHMS
R4=600 OHMS V_S=6.3 VOLTS

ANSWER? 65

I'm sorry but our answers don't agree.
I think your approach is incorrect.
Please TRY AGAIN or type 00 to review.
ANSWER? -1.9

You have the correct magnitude but the wrong polarity. Try Again.



CIRCUIT 7

FIGURE 6

QUESTION #1 FRAME 51

You will be given three chances to answer all questions correctly. If at any time you feel you would like to review the voltage divider technique concept, type "00" when a question is asked.

(The calculator mode may be entered by typing "$CA".) --Wait for the circuit.--

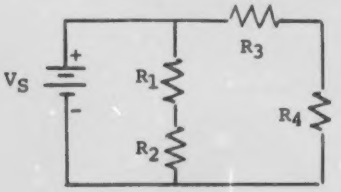
What is the value of the voltage drop across R2?

R1=1500 OHMS R2=670 OHMS R3=560 OHMS
R4=600 OHMS V_S=6.3 VOLTS

ANSWER? 65

I'm sorry but our answers don't agree. I think your approach is incorrect. Please TRY AGAIN or type 00 to review. ANSWER -1.9!

You have the correct magnitude but the wrong polarity. Try Again. LAST CHANCE ON THIS QUESTION ?1.9 GROOVY!



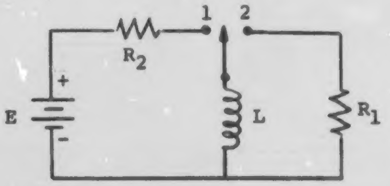
CIRCUIT 7

FIGURE 7

QUESTION #5 FRAME 8

TEST 1

Consider the following circuit.



Let E=10 VOLTS R1=3 OHMS
L=30 mH R2=6 OHMS

The switch has been in position 1 for a long time (at least 5 time constants). What is the current through the inductor?

ANSWER?

APPENDIX

You have missed the problem.

You have 3 choices as follows:

1. Type REVIEW for more drill.
2. Type CONTINUE if you would like to try the test problem again.
3. Type STOP to stop.

ANSWER?

Structured Programming

USE OF PROGRAMMING METHODOLOGY IN INTRODUCTORY COMPUTER SCIENCE COURSES

Elizabeth Alpert
Hartnell College
156 Homestead Ave.
Salinas, CA 93901
408 758-8211 x 431

INTRODUCTION

> "Twenty-five years ago our job was to instruct the machine; now it is the machine's job to execute our program."
> E.W. Dijkstra

In August 1979 I participated in the Ninth Annual Computer Science Institute at the University of California, Santa Cruz. The institute was dedicated to Programming Methodology--a one-week introductory course followed by a two-week lecture series. The speakers were an impressive group--many of the members of IFIP Working Group 2.3 plus several other prominent computer scientists. I expected to receive information I could use in the classroom, but I did not anticipate that the material would lead to a total reevaluation of what and how I was teaching.

The nature of computing is changing rapidly. The proliferation of microprocessors has spread the availability of computing and created an increased need for what Wirth (1) calls "programming know-how." In addition, advancements in microprocessors and LSI technology have paved the way for the development of new programming techniques. Economizing on memory space or eliminating an instruction or two are antiquated programming goals. The design and implementation of correct, readable, and understandable programs should be the goal of modern programmers.

Enormous contributions have been made toward reaching this goal. It is the responsibility of computer science educators not only to lecture about methodology but to incorporate the new methodologies thoughout the computer science curriculum. The purpose of this paper is to discuss those aspects of programming methodology that have particular relevance for computer science education and to make some suggestions for their incorporation early in the curriculum.

THE SCOPE OF PROGRAMMING METHODOLOGY

The major objective of programming methodology is to increase the programmers' ability to design and implement programs. Programming methodology addresses problem solving techniques, program reliability and adaptability, program correctness and structure, guidelines for partitioning large program tasks, and software tools. The computer science student, whether in a four- or two-year program, must develop an appreciation of what methodology is and how to use it.

PROGRAM DESIGN

Modularization

> "....a separation of concerns."
> E.W. Dijkstra

Most programming problems deal with more than one issue. Thinking about all the issues at once is confusing and difficult to do. By separating the issues and dealing with the complexities one by one, a program becomes naturally partitioned into modules. Programs written in a modular structure are easier to think about, modify, maintain, and understand.

How to partition a problem is not necessarily obvious, and therefore it is difficult to teach. What should be taught is that partitioning a problem is the first step in problem solving and program design, and is a worthy goal in itself.

Most introductory textbooks discuss modularization, but do not include enough exercises to provide students with an opportunity to apply it. It is up to the instructor to develop sophisticated problems that are simple to solve after the problem has been partitioned. The whole cycle of changing, adding, deleting, and exchanging modules should be made a part of all programming projects.

Specification

Specification is a statement of specific requirements. It is a process that is independent of composing the program and is not part of the solution. Specification provides a means of communication, and in fact should minimize what the programmer needs to know. To be useful, specifications must be precise and well-structured. Parnas (2) believes that specifications stated in terms of externally observable phenomena are preferable to specifications given as another program because they are user-oriented and understandable even by the non-programmer. They do not leave room for guessing about the requirements nor do they suggest particular implementations. There are various techniques of formal specification that appear in the programming methodology literature, but regardless of the method the common guidelines, as suggested by Parnas, are:

state everything that is required
state nothing that is not required
leave no room for doubt.

It is essential that students learn to appreciate the necessity of formal specifications and learn a well-structured specification methodology. Practical experience in writing formal specifications should be required of some programming projects. Most introductory textbooks provide completely specific problems. The students are even given the input/output formats. It would be a worthwhile exercise for instructors to assign practical problems with which the student has familiarity--e.g., registration processing, payroll--and require the student to do a complete specification of a problem.

ALGORITHM DESIGN AND DESCRIPTION

> "Nice descriptions of neat algorithms are not gifts from heaven; they are formally designed."
> Dijkstra and Gries

Once a program has been formally specified, algorithms can be designed for each of the parts. Before the algorithm is designed, however, it is necessary to select a notation to describe the algorithm. The notation must be concise, understandable, and formal.

An Introduction to Formal Descriptions

One of the most powerful tools for algorithmic design is the notation (or language) developed by Dijkstra (2). It is essentially a calculus for the derivation of programs using predicate transformers for each statement. A predicate transformer is a rule for deriving from a result predicate (post-condition) a predicate corresponding to the set of states that ensure that the statements will terminate and establish the truth of the post-condition. The statements in Dijkstra's language include: skip, abort, assignment, and the guarded constructs if---fi, do---od. A guard is a boolean expression at the head of a statement list; a list may be executed only if the corresponding guard is true.

The statements are defined by their predicate transformers. The mathematical semantics of a statement S and any post-condition R are given by the weakest pre-condition such that S establishes R. This is denoted by wp(S,R).

For the assignment statement the predicate transformer is

$$wp(\text{"}x := E\text{"}, R) = R_{x:=E}$$

where $R_{x:=E}$ denotes the predicate obtained by substituting all free occurrences of x in R by some expression E. A definition of the predicate transformers for all the statements can be found in (3). A description of the operational semantics for each statement follows.

The "skip" statement is executed by doing nothing. The "abort" statement signals failure. An assignment statement is denoted as "x := E" where x is a variable and E an expression. The semicolon denotes sequencing; the execution of "S_1; S_2" will result in S_1 being executed before S_2. The "IF" statement is denoted as

```
if  $B_1 \rightarrow SL_1$
 ▯  $B_2 \rightarrow SL_2$
 ▯  :
 ▯  $B_n \rightarrow SL_n$
fi
```

where the B_i denote guards and the SL_i are statement lists. In the execution of IF any one of the SL_i will be executed if the corresponding B_i is true.* The ▯ "bar" acts as a separator between otherwise unordered alternatives. The "DO" statement, denoted by

```
do  $B_1 \rightarrow SL_1$
 ▯  $B_2 \rightarrow SL_2$
 ▯  :
 ▯  $B_n \rightarrow SL_n$
od
```

continues to execute any one of the alternatives whose guard is true until all of the guards are false.

Some Examples of Formal Descriptions

Describing algorithms using this notation can be accomplished through some fairly simple formal methods. Let us consider a situation where it is desirable to establish the result as a = 7 where a is a program variable. The program a := 7 will establish the truth of the result since the initial state is true (i.e., 7 = 7 is T, and w("a := 7", a = 7) = T). The program a := 6 will not establish the truth of a = 7 since there is no initial state that will guarantee it (i.e., 7 = 6 is F) and wp("a := 5, a = 7) = F).

Let us next consider the problem of establishing, for a given x and y, a result where m, a program variable, contains the larger of x and y and either x or y when x = y. For the result to be true when m = x, x must be ≥ y. For the result to be true when m = y, y must be ≥ x. The conditions x ≥ y, y ≥ x become the guards for the program statements.

```
if $x \geq y \rightarrow m := x$

 ▯ $y \geq x \rightarrow m := y$

fi
```

Since one of the two conditions must be true the program will never abort. Note that if x = y both conditions will be true, and it is indeterminate which statement will be chosen; nor does it matter (3).

Programs using the do---od iterative construct can be developed in much the same way. Let us consider developing a program to establish a result R that states that S is the sum of the elements in an array A of n elements where n > 0. Obviously, the establishment of the truth of R requires a loop structure. What must be found is a generalization of R, i.e., a relation P, that can be easily made true initially and that can be held true during the iterative process. In generalizing, the variable i replaces the constant n. Thus, P can be defined to be the relation that S is the sum of all elements in the array up to A (i) and $0 \leq i \leq n$. Using this generalization, the result, R, is established if P is true and i = n.

Since P must be true before executing the loop, conditions must be found that establish this state. Setting i to 0 and S to 0 will establish P--the sum of no elements is zero. During the iterative process, computational progress must be made to ensure termination. Stepping i towards n by increments of 1 will do that. Keeping P invariant then requires adding A(i) to S after incrementing i. All that remains is the choice of the guard for the loop. Since i = n is the condition for termination, i ≠ n must be the condition for remaining in the loop. The program has now been derived:

```
S := 0; i := 0;

do $i \neq n \rightarrow$

   i := i + 1; S := S + A(i)

od
```

A more robust program results when the guard is weaker. A guard i ≠ n is weaker than i < n. If, for whatever reason, n were less than zero, the loop guard i < n would cause an immediate skip, and termination of the loop with no notification of error.

A common solution to this same problem is the following

```
S := A(1); i := 2

do $i \neq n + 1 \rightarrow$

   S := S + A(i); i := i + 1

od
```

---

*The IF aborts if no guard is true.

The second solution is more efficient--one pass through the loop has been eliminated. However, for n < 1 the second program is not correct. The program could be modified to the following equivalent program:

```
S := A(1); i := 1

do i ≠ n →

   i := i + 1; S = S + A(i)
od
```

but both will result in an infinite loop for n < 1.

Reasons for Using Formal Descriptions

By this time the reader might be thinking that the examples have not shown anything new. That is exactly the point. The solutions may be the same but they have been arrived at through a totally different process. It is the methodology that is important as a program design tool.

These solutions could probably have been written without the analysis, but for more sophisticated problems the program solution may not be as obvious or intuitive. And even if one were to arrive intuitively at the same program, that the development of the algorithm can be formally verified is significant.

By following the methods outlined by Dijkstra, solutions to problems can be devised and described in a notation that is quickly understood. The algorithms can be easily discussed because of the simplicity of the language. Initial formulations of algorithms can often be "massaged" producing simpler or more efficient solutions without changing the assertion about their validity.

Additional examples of this approach can be found in the appendix. It is suggested that the reader solve the problems first and then compare the solutions with the algorithms.

The greatest challenge in computer science education is teaching students how to design algorithms. The intuitive approach may work for some students but it certainly cannot be depended on. Illustrating the design of a few basic algorithms is usually helpful, because the students can transfer the techniques to related problems. For example, different algorithms for manipulating the elements in a list may share many of the same properties.

An initial reaction to the Dijkstra notation and the methods that are described much more formally in his book is that the methods are too mathematical and beyond the comprehension of most beginning students. That charge may be true for the underlying theory and the more sophisticated problems whose elegant solutions can only be understood when derived formally, but for the types of problems introduced at beginning levels the pre- and post-condition assertions as well as the iterative structures can be defined in a less formal manner. Dijkstra, himself in fact, often does just that.

What is important is that students be taught to design algorithms and be encouraged to express those designs in a notation that is independent of the programming language used for implementation. When algorithms are designed with a particular language in mind they are usually overly complicated and often incorrect. There is no great accomplishment in training students to think in FORTRAN (or any other programming language). The real goal should be to train students in how to think analytically and provide them with tools that can assist them in that task.

Other Formalisms

At this point the reader might be wondering why there has been no reference to flow charts, structured programming, or stepwise refinement. The design methodology that has been presented is actually an historical outgrowth of these techniques.

Twenty-five years ago, when programmers were concerned with the details of machine execution, flow charting might have been a useful technique. Now, however, it is antiquated as a design tool. The most appropriate use of the flow chart is as documentation--a description of the execution of the program. It is unreasonable to expect students to try to design algorithms using flow charts. Moreover, they hardly ever draw the flow charts before coding a program even when instructed to do so. They usually draw flow charts after the program has been coded and is running. Instead of being chastised for this, they should be praised.

Structured programming is more an implementation methodology than a design one. The constraints of structured programming for design purposes are fairly primitive. The Dijkstra methodology has equivalent basic constructs, and stronger guidelines for combining them. Students should certainly be taught structured implementation, but will find that implementing well-designed algorithms requires very little additional work.

Stepwise refinement still plays an important role in design, especially for larger problems. Dijkstra uses refinement in his notation when describing complex problems. Hehner (4) has suggested adding to the Dijkstra notation a formal method of refinement that eliminates the need for the do--od structure; and may further simplify the notation.

Students usually are successful when using refinement techniques. That an English phrase or descriptive term can be included in the design process(and its precise definition done separately), simplifies thinking about a problem. The practice of writing down simple concise steps is one that instructors should follow in the classroom whenever a problem is being worked out. Too often students attempt to solve problems by thinking in the syntax of the programming language when just a simple description suffices as a first step.

PROGRAM IMPLEMENTATION

> "Software is undoubtedly the major source of unreliability in most computer systems today."
>
> J.J. Horning

Unfortunately, the design of a correct algorithm does not ensure that its implementation will be reliable. Whereas the costs of hardware are plummeting, the costs of sortware keep rising. Seventy percent of programming effort is spent on maintenance and much of that maintenance is repair to unreliable or incorrect software. In view of this situation, one of the important goals of implementation should be reliability. This section of the paper presents some suggestions taken from Horning (5) for creating more reliable programs.

Language

The goals of simplicity, understandability, and maintainability are as important for program implementation as they are for program design. A simple, elegant algorithm is easier to implement than one that has not been carefully planned, but an effort must be made to maintain the simplicity. This effect can be achieved by choosing programming language constructs that are simple and easy to understand. Well-designed algorithms can be implemented using only a subset of a total language.

Another advantage of choosing a subset is that the programmer can become more familiar with the constructs and truly understand how to use them effectively. The use of a language structure that is only vaguely understood can be dangerous.

Very often in beginning programming language classes instructors cover as many of the language structures as possible. This is actually a disservice to the students. A more successful approach would be to throw out all the complex structures and consider only the very basic ones. The programmer who truly understands the basics will be more valuable than one who thinks he/she knows all the complexities.

Documentation and Style

Documentation and style are other areas that can improve reliability. Documentation should include not only a description of what the program is supposed to do but also information related to the design. Self-documenting techniques such as explicit declarations and choice of meaningful variable names are helpful.

Developing a style of programming should go right along with developing formalisms. Indentation and formatting of the program text is a relatively simple matter, yet it greatly enhances the reader's comprehension.

Most current textbooks provide examples of documentation and style. Without constant pressure from the instructor, however, there is little transfer of what the student sees in the text to the programming assignments.

Robustness

The concept of robustness was mentioned briefly in dealing with program design. It is relevant as well to program implementation. A program is more robust if it functions properly when given a wider range of input values. The inclusion of a few precautionary measures to handle the events that aren't supposed to occur can contribute much to the reliability of a program. These measures can be language, machine, or data directed. At present there are no exception handling schemes that are infallible, but nevertheless, it is vital that beginning programmers realize the importance of such considerations and develop an appreciation for the need to make programs robust.

CONCLUSION

> "Well it [program] works, it must be right."
>
> Anonymous

This paper has presented certain concerns of methodology in program design and implementation. These concerns should also be the concerns of computer science educators. The need for computer literacy courses is being stressed by college and university administrations across the nation. Soon every college student, whether in a two- or four-year curriculum will be enrolling in a computer science class. The question is--what and how are we going to teach these people?

For years, the philosophy has been that students need to get on the machine quickly. It didn't matter much what they did there. If we are teaching people how to program, it does matter. Many students take only one course in programming and use that experience in their future endeavors. With the increase in small business and home computers this pattern will increase. The decision must be made then whether these people should merely learn a programming language or really learn how to program.

The methodologies described in this paper are not difficult to understand or teach. They can be included in all elementary courses now being taught. If they are taught as part of the programming process so that students initially learn to do things correctly, the quality of programming is more likely to improve.

Teaching all the details of any programming language is of transitory value. Languages change, machines change, new versions of software are released, and no two manufacturers do everything the same. But teaching methodology has lasting value.

Just because a program works doesn't mean it's right. What must be conveyed to students is that there is more to programming than just coding. They need to develop an appreciation for programs that are simple, elegant, readable and robust--the kind of program of which Dijkstra says, "Ain't it a beauty!"

Acknowledgements

I would like to thank Professor William M. McKeeman and the University of California Extension for giving me the opportunity to participate in the Institute of Computer Science at UC Santa Cruz. I am grateful to Kenneth Friedenbach for his contributions to the development of this paper. I would also like to thank Professor Fred Schneider of Cornell University for his comments on an earlier version of this paper.

REFERENCES

*1. Wirth, Niklaus. The Module: A System Structuring Facility in Higher-Level Programming Languages. Institut fur Informatik, Zurich.

*2. Parnas, David L. The Role of Program Specifications. University of North Carolina at Chapel Hill.

3. Dijkstra, Edsger W. A Discipline of Programming. Prentice-Hall, Inc. 1976.

*4. Hehner, Eric C.R. do Considered od; A Contribution to the Programming Calculus. University of Toronto.

*5. Horning, J. J. Effects of Programming Languages on Reliability. Xerox Palo Alto Research Center.

APPENDIX

The following are examples of algorithms developed by formal methods presented by David Gries during a five-day Introduction to Programming Methodology class at the University of California, Santa Cruz, August 1979.

Example 1. Coincidence problem

Given a function f with M values where

$$f(0) < f(1) < f(2) < ----- < f(M-1)$$

and a function g when N values where

$$g(0) < g(1) < g(2) < ----- < g(N-1)$$

count the pairs that are the same. For example, for

f = 3, 5, 8, 12, 15, 18
g = 1, 3, 4, 7, 8, 11

the count c would be 2.

```
Solution 1.  m := 0; n := 0; c := 0;

     do n ≠ N and m ≠ M →

         if f(m) < g(n) → m := m + 1

          ▯ f(m) > g(n) → n := n + 1

          ▯ f(m) = g(n) → n := n + 1;
                          m := m + 1;
                          c := c + 1

         fi
     od
```

The loop invariant is that c contains a count of the number of f(i) = g(i) where $0 \le i < m$, $0 \le j < n$. Termination is the condition $m = M$, provided $f(m-1) < g(n)$ or $n = N$, provided $g(n-1) < f(n)$.

---

*These were distributed at the Programming Methodology Lecture Series, Ninth Annual Computer Science Institute, University of California, Santa Cruz, August 1979.

Example 2. Different Values
Given a function f where

$f(1) \leq f(2) \leq f(3) \leq \text{-----} \leq f(M),\ M \geq 1,$

count the number of different values in f(1:M).

Solution 2.

```
m := 1; c := 1

do m ≠ M →

   if f(m) = f(m + 1) → m := m + 1

    □ f(m) ≠ f(m + 1) → c := c + 1;
                        m := m + 1
   fi
od
```

The loop invariant is that c contains 1 + the number of $f(i - 1) \neq f(i)$ for $1 < i \leq m$ and $1 \leq m \leq m$.

The program may be simplified to:

```
m := 1; c := 1

do m ≠ M →

        m := m + 1
        if f(m) = f(m + 1) → "skip"
         □ f(m) ≠ f(m + 1) → c := c +1

        fi
od
```

FORTRAN 77: Impact on Introductory Courses in Programming Using FORTRAN

by

Frank L. Friedman
Department of Computer and Information Sciences
Room 381 Speakman Hall
Temple University
Philadelphia, Pennsylvania 19122
(215) 787-1912

ABSTRACT

A first course in teaching problem solving and structured programming using FORTRAN is briefly described. The features of the new FORTRAN 77 standard which in the author's view impact most significantly upon the course are summarized, and the effect of those features upon the structure and content of the course is discussed.

INTRODUCTION

On the third of April, 1978, The American National Standards Institute (ANSI) approved a new American National Standard for the FORTRAN programming language [ANSI 78]. This standard, designated as FORTRAN 77, is a revision of the 1966 American National Standard FORTRAN. It is expected to have a significant impact upon the use of the FORTRAN language in a wide variety of applications areas. The new language should also have a considerable effect upon the use of FORTRAN as a convenient language for teaching introductory programming, since it provides a number of significant pedagogic advantages over its predecessor.

The features of FORTRAN 77 that support these advantages are the subject of this paper. A summary description, with examples, of each feature is described, and the effect of the feature upon the structure and content of an introductory course is discussed. Before proceeding to these topics, however, an outline description of such a course is presented (see also [FK 77 and FK 78]).

COURSE STRUCTURE

Figure 1 contains an outline of the topics covered in the course, a time scale for these topics, and a list of problems that are associated with each topic. The course is oriented around a set of two-dozen problems which illustrate a variety of problem-solving techniques. Most of these problems are solved in their entirety, from the analysis and initial algorithm outline, through to the final flow-diagram refinements and FORTRAN program.

Each problem is used to illustrate the application of a new feature of the FORTRAN language. The feature will normally have been introduced first, and a brief description of its syntactic form provided. The problem provides additional motivation for mastering this new feature since the problem solution would be much more difficult without it.

Some of the problems shown in Fig. 1 emphasize skills that are fundamental to programming, such as finding the largest value in a data collection, searching an array for a specified item, and sorting. Other problems relate to a variety of application areas: business-oriented problems (checking account transactions and inventory control), games (bowling score computation and Tic-Tac-Toe), statistical computations, computer graphics, and text editing.

By concentrating on problems that require the introduction of additional features of the FORTRAN language for a reasonable solution, the motivation for these features becomes readily apparent. Once the essentials of the features needed to solve a particular problem are introduced, class discussion focuses on data description and algorithm design. It is occasionally the case that other FORTRAN features are introduced as the algorithm is developed, refined, and finally implemented. The essentials of these features are described, and examples of their use are given, usually within the context of the problem at hand.

SUMMARY OF NEW FEATURES

The new FORTRAN standard describes two levels of the language: FORTRAN (sometimes referred to as Full FORTRAN), and Subset FORTRAN. Whereas the FORTRAN subset was previously described in a separate standard (American National Standard Basic FORTRAN, ANSI X3.10-1966), the description of Subset FORTRAN is now included in the description of the full language, and the old standard has been withdrawn.

The guiding criteria used in the development of the standard were [BRAI 78]:

1. the inclusion of only those new features proven through actual usage
2. the inclusion of new features that enhance the portability of programs

Week		
1-2	Introduction to computers, programs and programming languages FORTRAN and the basic computer operations	Computation of gross and net salary for one person (the program given to students to run on the computer during the first week)
3	Problem Analysis Algorithm development and refinement Flow diagrams	Computation of sum and average of N items
4	One and two alternative decisions WHILE loops Compiler role in transrelating structures	Inventory control, finding largest number, simulation of a bicycle race
5	Data types in FORTRAN List-directed formatting DO loops Arithmetic expressions and functions	Checking account program, prime number identification, Centigrade to Fahrenheit conversion
6-7	Lists and subscripted variables Searching a list Index computation	Computation of table of factorials, finding an item in a list, frequency distribution of exam scores. Table lookup via direct computation and search arrays
8	Block-IF decisions structure Generalized DO loop Next iteration and loop exit control Nested Structures	Scoring a bowling game, drawing a bar graph, sorting an array
9-10	Functions and subroutines Argument lists and global data Program system charts	Simple statistical package Sort/merge package
11	Use of Format Statements	Mortgage interest tables
12-13	Logical expressions Character string processing Extracting substringe Replacing substrings	General search subroutine, finding parameters of a DO loop header, text editor program system
14	Multi-dimensional arrays Array input and output Computer art	Matrix inversion, status of a Tic-Tac-Toe game, printing block-letter patterns, scheduling class rooms

Fig. 1: Course Outline and Assigned Problems

3. minimal increase in language or processor complexity
4. avoidance of features that conflict with the previous (1966) standard
5. elimination of features in the 1966 standard only under clearly demonstrated circumstances
6. production of a more precise description of the language.

The new standard describes programs written in FORTRAN 77, and not the processors (such as a compiler or interpreter) of these programs. The implementation of a standard-conforming processor is to be inferred from the standard. The standard is to be interpreted as specifying only the minimum requirements of the language. Thus a standard-conforming processor for the language must be able to handle all standard-conforming programs according to the rules of the standard. It may, in addition, however, have extensions for features such as bit manipulation or array processing that are not specified in the language. It is then the decision of the user whether or not to conform to the standard when writing a program. Of course, standard-conforming programs usually will be portable to all machines supporting a standard-conforming compiler; non-standard programs may not be as portable.

This section contains a list and description of the new features of Full FORTRAN which affect most significantly the introductory course just described. Those features discussed that have not been included in the subset are so designated. Users of systems not supporting the full language should make adjustments as appropriate in the curriculum changes suggested in the last section of the paper.

The relevant features of Full FORTRAN are listed in Table 1 and summarized in the remainder of this section. The material presented is in no way intended as a complete description of these features; in fact, it barely scratches the surface. Additional examples and detail may be found in the Brainerd paper and in FORTRAN 77 introductory programming texts (see [DH 78], and [HH 79]).

TABLE 1

NEW FORTRAN 77 FEATURES MOST RELEVANT TO INSTRUCTION IN INTRODUCTORY PROGRAMMING

1. The character data type
2. List-directed formatting (not in the subset
3. The block-IF (if-then-else) decision structure
4. Generalized form of the DO Statement
5. Arrays
6. Expressions: the PARAMETER Statement and mixed-mode arithmetic
7. The SAVE statement.

1-The Character Data Type

Perhaps the most significant change in the standard is the addition of the character data type, which now replaces the Hollerith type. It was the use of the Hollerith type that made many FORTRAN programs difficult to understand, to check out, and to transport from one computer to another having a different size storage cell. (Although the Hollerith type is no longer included in the standard, it is expected that most major manufacturers will continue to support this feature in their FORTRAN 77 processors).

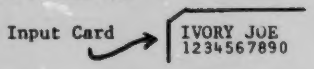
Some examples of the declarations of character variables and arrays are:

```
CHARACTER*120 BUFFER
CHARACTER CARD (80), ITEM
CHARACTER*10 FNAME, INITLS *1, LNAME
```

The length of each character variable and array is fixed by the declaration statement. In this example, BUFFER is declared as a character string of length 120, while CARD is taken to be an 80 element array of character strings of length 1.

Character constants consist of strings of characters enclosed in apostrophes. Character variables and array elements may be given values in the same manner as other typed elements in FORTRAN: through the use of READ, assignment, and DATA statements. For example, the statements shown below all have the effect of assigning the string IVORY JOE to the variable FNAME, as previously declared.

```
   READ (FMT=10, UNIT=5) FNAME
10 FORMAT (A)
```



```
FNAME = 'IVORY'  //  'JOE'
FNAME = 'IVORY JOE'
DATA FNAME  /  'IVORY JOE'  /
```

The READ statement illustrates the use of specifiers for format and unit numbers. Other input/output specifiers (for end-of-file, error conditions, etc.) are also allowed (see [MO 80]). The old form

```
READ (10, 5) FNAME
```

is still permitted with the expected caveat that the first item listed specifies the unit number and the second, the format number. The use of FMT and UNIT have the added advantage of allowing order-independent list specifiers in an input/output statement.

The use of the A descriptor by itself in a format is also new to FORTRAN. When the length of the element to be transmitted is not specified in the format, [illegible] is taken from the declared length.

The statement

```
FNAME = 'IVORY'  //  'JOE'
```

illustrates the use of the character string concatenation operation. Substring and string comparison operations are also permitted in FORTRAN 77, as well as intrinsic function operations for determining string length and for character-to-integer and integer-to-character conversion. User-defined character functions are also permitted in FORTRAN 77. (The FORTRAN 77 Subset does not support concantenation, substrings, or character functions).

2-List-Directed Formatting (Not available in the Subset)

The new FORTRAN standard permits the use of a statement label, an integer variable that has been assigned a label, a character expression, or an asterisk for the designation of a format. For example, statement lists i), ii), iii) and iv) are allowed in FORTRAN 77 and produce identical results.

```
i)   WRITE (6, 150) 'SUM = ', N * (N + 1) / 2
 150 FORMAT (A, I6)

ii)  ASSIGN 150 TO LABEL
     WRITE (6, LABEL) 'SUM = ', N * (N + 1) / 2
 150 FORMAT (A, I6)

iii) CHARACTER FORM*6
     FORM = '(A, I6)'
     WRITE (6, FORM) 'SUM = ', N * (N + 1) / 2

iv)  WRITE (6,'(A, 16)' 'SUM = ', N * (N + 1) / 2
```

An asterisk is used to specify list-directed formatting, as determined by the input/output list, the processor, and the form of the data. For input, the type of each data item is determined by the FORTRAN system (rather than a user format description). Additional information, such as the position of the decimal point in a real number is also determined in this manner. Data items may be separated from one another using blanks or commas.

Given the card

```
'219-40-0677'  'LITTLE LOOT'  80  1.35
```

The statements

```
REAL RATE
CHARACTER SSNO*11, NAME*24
INTEGER HOURS
READ*, SSNO, NAME, HOURS, RATE
```

would produce the following result:

SSNO	NAME	HOURS	RATE
219-40-0677	LITTLE TOOT	80	1.35

As shown, character strings read via list-directed formatting must be enclosed in apostrophes; the four data items shown in the card are separated from one another by one or more blanks.

In list-directed output, the widths of the fields used to print the listed elements are determined by the type of the element and the processor. The width of each character string to be printed is determined by the declared length of the string. However, real and integer data are printed in fixed-width fields regardless of the magnitude of the value to be printed; the field widths are pre-determined by the processor and are normally not alterable by the programmer.

3-The Block-IF Decision Structure

The block-IF structure will considerably reduce the reliance upon the GOTO and the label in FORTRAN programming. This structure, with appropriate code indentation, can greatly enhance the readibility of programs written in FORTRAN 77.

The logical function PRIME shown in Fig. 2 (see [BRAI 78] provides a good illustration of the use and advantages of the block-IF.

```
   LOGICAL FUNCTION PRIME
   INTEGER N, DIVISR
   IF (N .LE. 1) THEN
      PRIME = .FALSE.
   ELSE IF (N .EQ. 2) THEN
      PRIME = .TRUE.
   ELSE IF (MOD(N, 2) .EQ. 0) THEN
      PRIME = .FALSE.
   ELSE
      DO 10 DIVISR = 3, INT(SQRT(REAL(N))), 2
         IF (MOD (N, DIVISR) .EQ. 0) THEN
            PRIME = .FALSE.
            RETURN
         ENDIF
10    CONTINUE
      PRIME = .TRUE.
   ENDIF
   RETURN
   END
```

Fig. 2: An Illustration of the Block-If Structure

This function shows the use of the Block-IF in implementing a decision structure with four alternatives, and a decision structure having one alternative (inside loop 10). A two-alternative Block-If may be implemented in the form

```
IF (condition) THEN
   -  }  if condition is true statement
   -  }  sequence executed
ELSE
   -  }  if condition is false statement
   -  }  sequence executed
ENDIF
```

4-Generalized Form of the DO loop Statement

The previous sample also illustrates the more general DO loop feature provided in FORTRAN 77. The form of the FORTRAN 77 DO loop header statement is:

```
DO sn loopvar = exp1, exp2, exp3
```

where

i) exp1, exp2, and exp3 represent the initial, terminal, and step value expressions respectively

ii) exp1, exp2, and exp3 may be any integer, real, or double precision expression having positive negative, or (except for exp3) zero values (the use of expressions is not permitted in the Subset)

iii) Loopvar may be any integer, real, or double precision variable

iv) an optimal comma is permitted after the terminal statement label, sn.

The number of times a DO loop is executed is called the trip count. The trip count specified by the previously described loop header is computed as

```
MAX(INT(v2-vl + v3)/v3),0)
```

where vl, v2, and v3 are the values of the expressions expl, exp2, and exp3 respectively, and INT truncates the result of the expression argument if it is not an integer value. The following relationships must hold between v,l, v,2 and v3:

$vl < v2$ and $v3 > 0$

or

$vl > v2$ and $v3 < 0$

If the value of the trip count is not positive, the loop will not execute at all. It is important to note that this is contrary to the convention adopted by many processors based upon the previous standard: that loops were executed at least once regardless of the values of the loop parameters. The previous standard did not specify what was to be done for loops written with an initial value exceeding the terminal value at loop entry. The change will not effect the execution of programs that were written in accordance with the previous standard, but it may affect those that were written in violation of the standard.

5-Arrays

The Full FORTRAN language defined in the standard provides three major changes from the previous standard:

i) arrays may have up to seven dimensions
ii) the specification of a lower subscript bound is allowed
iii) subscripts in an array reference may be any integer expression (see also, section 6).

Items i) and ii) above, are not permitted in the Subset.

The specification of the lower subscript bound is indicated through the use of a colon, as in

```
REAL X (-3:5)
```

which defines a nine-element array X with elements X(-3), X(-2)...X(0)...X(5). If the lower bound is omitted, it is assumed to be one.

6-Expressions: The PARAMETER Statement and Mixed Mode Arithmetic

There are several places in the new FORTRAN in which expressions are permitted where only constants, variables, or restricted forms of expressions were previously allowed. For example, expressions are now allowed in output statements (see i. below), as subscripts (ii), as array dimension bounds (ii), and as indexed-DO parameters (see section 4). Some examples are

```
  i)  WRITE (6, 150) 'SUM = ', N * (N + 1) / 2
 ii)  INTEGER SIZE, I, J, K
      PARAMETER (SIZE = 10)
      REAL X(2*SIZE)
      DATA (X(I), I = 1, SIZE) / SIZE*0 /
      WRITE (6, *)  X(6*K-J)
```

Example ii) illustrates the use of the PARAMETER statement, which is used to attach a symbolic name to a program constant or parameter. (The PARAMETER statement is not included in the Subset). If the symbolic name is not of the default implied type, its type must be specified before its appearance in a PARAMETER statement. Expressions are permitted to the right of the equal symbol, but they may contain only constants or previously defined symbolic constants.

Symbolic constants may be used in expressions in the same way as variables. They may, in addition, be used in specification and DATA statements in places where only constants had been previously allowed. Such uses are illustrated in lines 3 and 4 of Example ii), where the symbolic constant SIZE is used where only constants were previously allowed.

Example ii) also illustrates the use of the implied-DO in a DATA statement (not allowed in the Subset) and the generalization of the form of subscripts allowed in FORTRAN 77. Any integer expression may be used to specify a subscript, provided the value is within the bounds specified for the corresponding dimension in the array declaration. (The old standard limited subscript specification to expressions of the form

$$C_1 * V \pm C_2$$

where C_1, C_2 were integer constants, and V was an integer variable).

With regard to the formation of expressions, the major difference between the 1966 standard and the new FORTRAN is the inclusion of mixed-mode arithmetic. Integer, real, double precision, and complex operands may appear in arithmetic expressions except that double precision and complex operands may not appear in the same expression.

The type of an expression is determined by examining operand-operator-operand triples—the type of the result of each triple is determined by the type of the two operands involved. For example, if K, I, and J are integers, and I=5, J=2, then the result of the evaluation of the statement

```
K - 2.5 + I/J
```

is 4, as determined as follows:

i) divide I/J; since I=5 and J=2 are both integers, the result of this division is an integer, 2.

ii) add the real value 2.5 and the integer 2; since 2.5 is real, the result of the division is first converted to real prior to addition which then yields a result of 4.5.

iii) the real result 4.5 is assigned for the integer variable K, resulting in the actual assignment of the truncated value 4.

7-The SAVE Statement

Contrary to what a number of FORTRAN users have become accustomed to, there is no requirement in either the 1966 standard or the new standard to retain the values of local variables in subprograms from one execution to the next. Programs written under the assumption that local variables were saved were non-standard and would not execute correctly on all processors.

While saving the value of local variables in subprograms used to be rather cumbersome, the FORTRAN 77 SAVE statement can be used to simplify this process. For example, a small subprogram to increment a counter and check for overflow might appear as follows:

```
SUBROUTINE BUMPIT
INTEGER MAXVAL
PARAMETER (MAXVAL = 100)
INTEGER COUNT
DATA COUNT /0/
SAVE COUNT
COUNT = COUNT + 1
IF (COUNT .GE. MAXVAL) THEN
   PRINT*, 'COUNTER EXCEEDS MAX VALUE OF ', MAXVAL
   PRINT*, 'COUNTER RESET TO 1.'
   COUNT = 1
   RETURN
ENDIF
RETURN
END
```

The SAVE statement causes the value of COUNT to be saved between calls to the subprogram BUMPIT.

IMPACT OF NEW FEATURES UPON INSTRUCTION

Only a few of the new features found in FORTRAN 77 have been described in the previous section. As indicated, the list is restricted to those features which are felt to have the greatest impact upon instruction in introductory programming courses using FORTRAN. Yet of the features discussed, only three, the character data type, the Block-IF structure, and list-directed formatting have substantially contributed to major changes in the structure and content of the course. Of the remaining features, the generalized DO loop (introduced in week 8), implied-DO in a DATA statement (weeks 6-7), additional flexibility in the use of expressions (spread throughout the course), and the additional array features (week 8) are creatures of increased convenience having only a minor impact upon the course. The SAVE statement (weeks 9-10) is a feature that should be understood by all students working with subprograms, but it has no other impact upon the course.

The PARAMETER statement (introduced in week 4) provides a vehicle for discussing the notion of a program parameter (as opposed to an in-line constant). This statement provides a convenient means for a programmer to attach a name to a constant value (programmer parameter) that has special program significance. A value representing the maximum size of an array is one example of a program parameter. Other examples are illustrated in the sample program shown at the end of the paper.

In the view of this author, mixed-mode arithmetic offers no advantage to the student in an introductory course. On the contrary, the use of mixed-mode expressions requires additional care and a more sophisticated understanding of expression evaluation than is desirable or even necessary at this level. The automatic type conversion required by the mixed-mode expression provides very little programming convenience and has the added disadvantage of being hidden from the user. The use of the type conversion functions such as INT (real-to-integer with truncation), AINT (real-to-nearest integer), and REAL (integer-to-real) in avoiding mixed-mode arithmetic should be encouraged.

It is perhaps not too surprising that the character data type, Block-IF, and list-directed formatting have had the most influence upon the course. For, as an analysis of the course structure outline indicates, the major emphasis in the course is upon problem solving, rather than the details of the FORTRAN language. These three features make it even easier to concentrate on problem-solving techniques and algorithm development, with less emphasis upon the language implementation considerations. Yet each contributes to this effort in a different way.

The character data type makes it possible to introduce the concept of a character string, and the reading, printing, and comparison of strings at a very early stage of the course. For example, character string constants are used in list-directed output statements as early as week 1, in order to provide descriptive labels or headers for the values printed by the first program run by a student. By week 4, the student is reading and printing character strings using list-directed formatting and simple character string comparison. During week 4, data types are discussed in detail, and a more formal presentation of the character type is given. Finally, by weeks 12 and 13, students are writing programs requiring the use of some simple but fundamental string operations such as substring extraction, comparison, insertion, deletion, and replacement. Most important, all work is now done in a totally machine-independent fashion, without the previous requirement of having to store data of one type (character) in a memory cell of a different type (real or integer). This latter feature has the added advantage of providing additional compile-time checking for data type and operator consistency.

The Block-IF structure eliminates most of the need for GOTOs and labels in the implementation of algorithms in FORTRAN 77. The only remaining GOTO needs now can easily be restricted to implementation of the WHILE loop structure and loop exit and next iteration steps. The Block-IF makes it possible for instructors and students to concentrate on the specification of decision steps in terms of the tasks to be accomplished and the condition(s) of selection, without concern for the details of using labels and GOTOs. The resulting implementation is not only easier to write, but also easier to understand and far less subject to error. This point is perhaps best illustrated via one possible rewrite using GOTOs and labels of the executable portion of the function PRIME (see Fig. 3).

```
      IF (N .GT. 1) GO TO 1
        PRIME = .FALSE.
        RETURN
    1 IF (N .NE. 2) GO TO 2
        PRIME
        RETURN
    2 IF (MOD(N, 2) .NE. 0) GO TO 3
        PRIME = .FALSE.
        RETURN
    3 MAX = IFIX(SQRT(FLOAT(N)))
        DO 10 DIVISR = 3, MAX, 2
          IF (MOD(N, DIVISR) .NE. 0) GO TO 10
            PRIME = .FALSE.
            RETURN
   10   CONTINUE
        PRIME = .TRUE.
        RETURN
```

Fig.3: The Function PRIME Without the Block-IF

Even with indentation, the logical structure of this code is obfuscated considerably because of the GOTOS and labels.

The single and double alternative forms of the block-IF are introduced during week 3 of the course, and the general form is presented during week 8. By this time, however, students have been doing considerable programming, using the GOTO only for the implementation of the WHILE loop structure. (Even this use could have been avoided had the FORTRAN 77 standard contained a WHILE-like loop structure in which the repetition condition could be specified in the structure header).

Another advantage of the block-IF is increased similarity in the structure of student programs. Students (and instructors, too) are now better able to understand the programs of others and to assist in checking out and correcting programs that fail.

Finally, there is the matter of list-directed formatting. List-directed formatting allows the instructor to delay considerably any discussion of one of the most detailed, unpleasant features of the FORTRAN language -- formats. While it is true that formats are an important feature of the FORTRAN language, and should not be overlooked, the study of formats contributes little to student mastery of the fundamentals of choosing data structures and designing and implementing algorithms. It is for this reason that formats have been delayed until week 11 of the course, after the presentation of subroutines and functions is complete. It is indeed a shame that the list-directed formatting feature is not included in the Subset. It is hoped that most Subset implementors will view it as top priority for support in their processors.

In the course, list-directed formatting is introduced in the first program given to students to prepare and submit for computer entry. The capabilities of the list-directed feature are then slowly expanded informally throughout weeks 2 through 4 until finally, in week 5, it is formally described, and a brief overview of how it works is presented. Exclusive use of list-directed formatting in all problems studied and assigned is then continued until week 11.

CONCLUDING COMMENTS

Of all the new features of FORTRAN 77, list-directed formatting provides the greatest added convenience to instructors of introductory programming courses (be they in FORTRAN, BASIC, PASCAL, or any other language). The block-IF and character features are not far behind in this respect. But list-directed formatting allows students to begin to do input and output immediately, and to continue to do even slightly sophisticated input/output throughout the first ten weeks of the course (including the study of subprograms) without the added complication introduced through the use of formats.

The following program (Fig. 4) illustrates the use of list-directed formatting, the character data type, the block-IF structure, and the PARAMETER statement. Arrays are used in the program only for illustration, not because they are required. The program was run on a Control Data Corporation Cyber 174 using the University of Minnesota FORTRAN compiler, M77. For the example input entries show below

```
    6
    'Big Bird' 5
    'Mickey Mouse' 2
    'Kermit T. Frog' 6
    'Ace Bandage' 95
    'Carmina Burana' 7
    'Miss Piggy' 9
```

the generated program output is:

```
NUMBER OF EXAM SCORES IS 6
    NAME                SCORE        RATING
BIG BIRD                  5     SATISFACTORY
MICKEY MOUSE              2     UNSATISFACTORY
KERMIT T. FROG            6     SATISFACTORY
ACE BANDAGE              95     *** INVALID SCORE **
CARMINA BURANA            7     SATISFACTORY
MISS PIGGY                9     OUTSTANDING

THE NUMBER OF OUTSTANDING SCORES IS 1
THE NUMBER OF SATISFACTORY SCORES IS 3
THE NUMBER OF UNSATISFACTORY SCORES IS 1

THE NUMBER OF INVALID SCORES IS 1
```

REFERENCES

[ANSI 78] American National Standard Programming Language FORTRAN, American National Standards Institute, New York, 1978.

[BRAI 78] Brainard, Walter S. et. al., "FORTRAN 77", CACM (21, 10), October 1978, pp. 806-20.

```
C  PROGRAM TO PROCESS A SET OF EXAM SCORES RANGING BETWEEN 0 AND 10
C  AND CLASSIFY ACCORDING TO OUTSTANDING (8-10), SATISFACTORY (4-7),
C  OR POOR (0-3).  COMPUTE AND PRINT FREQUENCY COUNTS.
C
C  FRANK L. FRIEDMAN     12-9-79
C
       INTEGER MINOUT, MINSAT, MINSCR, MAXSCR
       PARAMETER (MINOUT = 8, MINSAT = 4, MINSCR = 0, MAXSCR = 10)
       INTEGER MAXCNT
       PARAMETER (MAXCNT = 100)
       CHARACTER*24 NAME (MAXCNT)
       INTEGER SCORE (MAXCNT), N, II, I
       CHARACTER*20 RATING
       INTEGER OUTNR, SATNR, UNSNR, ERRCNT
C
C  READ AND VALIDATE N
       READ*, N
       IF (N .GT. 0 .AND. N .LE. MAXCNT) THEN
          PRINT*, 'NUMBER OF EXAM SCORES IS ', N
       ELSE
          PRINT*, 'NUMBER OF EXAM SCORES IS NOT VALID, MAX IS ', MAXCNT
          PRINT*, 'PROGRAM TERMINATED.'
          STOP
       ENDIF
C
C      READ DATA
          DO 20 II = 1, N
            READ*, NAME(II), SCORE (II)
       20 CONTINUE
          PRINT*, ' '
          PRINT*,            NAME           SCORE           RATING'
C
C  PROCESS EACH RECORD.  DETERMINE AND PRINT RATING ALONG WITH
C  NAME AND SCORE.  INCREMENT APPROPRIATE COUNTER.
       OUTNR = 0
       SATNR = 0
       UNSNR = 0
       ERRCNT = 0
       DO 40 I = 1, N
          IF (SCORE(I) .LT. MINSCR .OR. SCORE (I) .GT. MAXSCR) THEN
            RATING = '*** INVALID SCORE ***'
            ERRCNT = ERRCNT + 1
          ELSE IF (SCORE (I) .GE. MINOUT) THEN
            RATING = 'OUTSTANDING'
            OUTNR = OUTNR +1
          ELSE IF (SCORE (I) .GE. MINSAT) THEN
            RATING = 'SATISFACTORY'
            SATNR = SATNR + 1
          ELSE
            RATING = 'UNSATISFACTORY'
            UNSNR = UNSNR + 1
          ENDIF
          PRINT*, NAME(I), SCORE (I), '  ', RATING
   40 CONTINUE
C  PRINT COUNTS
       PRINT*, ' '
       PRINT*, ' THE NUMBER OF OUTSTANDING SCORES IS ", OUTNR
       PRINT*, ' THE NUMBER OF SATISFACTORY SCORES IS ', SATNR
       PRINT*, ' THE NUMBER OF UNSATISFACTORY SCORES IS ', UNSNR
       PRINT*, ' '
       PRINT*, ' THE NUMBER OF INVALID SCORES IS ', ERRCNT
C
       STOP
       END
```

Fig. 4: Sample FORTRAN 77 Program

[DH 78] Davis, Gordon B., and Thomas R. Hoffman, FORTRAN: A Structured, Disciplined Style, McGraw-Hill, 1978.

[FK 77] Friedman, Frank L. and Elliot B. Koffman, Problem Solving and Structured Programming in FORTRAN, Addison-Wesley, 1977.

[FK 78] Friedman, Frank L. and Elliot B. Koffman, "Teaching Problem Solving and Structured Programming in FORTRAN," Computers and Education (2, 3), Pergamon Press, January 1978, pp. 235-45.

[HH 79] Hume, J.N.P., and R.C. Holt, Programming FORTRAN 77: A Structured Approach, Reston, 1979.

[MO 80] Meissner, Loren P. and Elliot I. Organick, FORTRAN 77 Featuring Structure Programming, Addison-Wesley, 1980.

USING MODEL-BASED INSTRUCTION TO TEACH PASCAL

Bogdan Czejdo

Warsaw Technical Univ.
Brigham Young Univ.

I. INTRODUCTION

In this paper we introduce the concept of model-based instruction, first defining "model" and then presenting a simplified view of the modeling process. A way of classifying models is presented, and a process for preparing a set of complementary models for use in model-based instruction along with the characteristics of this type of instruction. The paper then concludes with the application of model-based instruction to computer-assisted instruction.

II. THE ROLE OF MODELS IN TEACHING

A model is defined in Webster's Dictionary(1) to be a*:

1. copy, image
2. pattern of something to be made
3. archetype
4. description or analogy used to help visualize something that cannot be directly visualized
5. system of postulates, data, and inferences presented as a mathematical description of an entity or state of affairs

For the purposes of this paper, a model is defined to be any diagram, table, picture, or figure which helps the student to understand and remember a concept or perform an action which is a part of a set of teaching objectives. As shown by research in a variety of teaching areas,(2) models play a very important role in the learning process.

A simplified view of the modeling process in learning is represented in Figure 1.

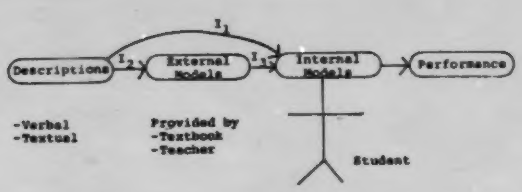


Figure 1. The Creation of Models

The internal models students created in their own minds allow them to understand and remember concepts and perform prescribed actions. The students may create these models on their own (transformation I_1) based on descriptions provided in textbooks and lectures. Alternatively, the internal models used by the student may be based on external models (transformation I_3) already created by the textbook author or lecturer (transformation I_2).

The creation and use of internal models is a complex psychological process which will be mentioned only briefly here. It should be recognized that the creation of models by the student is a difficult and time-consuming task. During lecture, there is usually insufficient time for the student to formulate suitable models on his own. The fuzzy and incomplete models that may come to mind are soon forgotten because they are not strongly impressed on him through reinforcement by the teacher or by his own performance.

Indeed, transformation I_1 is difficult for the student for a variety of reasons. The student may be untrained in the creation of models or have a weak imagination. Because of an incomplete understanding of the full subject, the models he creates may turn out to be unsuitable. Unlearning an initial model and relearning a more adequate model may be very difficult.

---

* We have chosen a subset of meaning which corresponds with the meaning of model in our paper.

For all of these reasons, it is important for the teacher, and in particular the computer science teacher, to pay careful attention to the subject of models and to provide the student with useful external models. (In the rest of this paper, the term "model" will refer to external model.)

III. TYPES OF MODELS

A wide variety of models have been examined and evaluated for potential use in teaching PASCAL programming. These models have been drawn from several textbooks(3,5) and have also been created by the authors of this paper. What constitutes the model and the object to be modeled depends on the point of view of teacher and student. A computer program, for example, is often a model of some real-world activity or process. However, in the context of this paper, the computer program will be considered to be the object to be modeled.

One of the best ways to classify models is by their structure. Six basic structures have been identified:

- graphic
- array
- mathematical
- text
- compound
- parallel

Most of the models we use are graphic structures, four of which are levels, trees (hierarchies), networks, and domains. Levels and tree structures can be described using the example of an interactive language. On the microcomputer systems used by the authors in their introductory PASCAL course, there is an interactive command language which is interpreted by a system monitor or operating system. Each user starts at the command level. By typing an "E", the user can go to the edit level. To return to the command level from the edit level, the user can type a "QU" sequence. Figure 2 shows simple level structure.

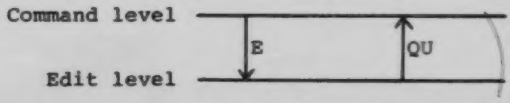


Figure 2. Simple Level Structure

Levels are represented by horizontal lines. The transitions are represented by labeled vertical lines. Figure 3 shows a simple tree structure.

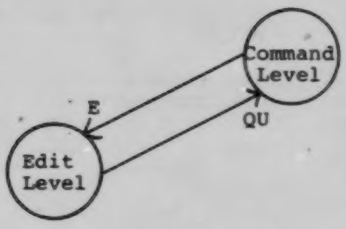


Figure 3. Simple Tree Structure

Each circle represents being in a particular state (at a particular level in this case).

It should now be obvious how one can combine elementary submodels into a more complex structure. In Figures 4 and 5, a more complete model of an interactive language is drawn using the level and tree structures of Figures 2 and 3.

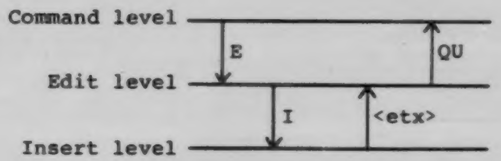


Figure 4. Level Structure

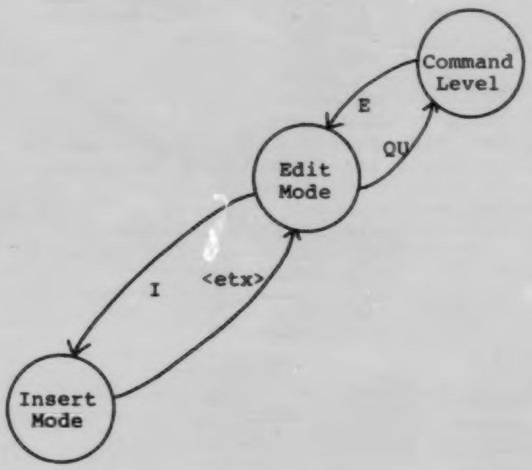


Figure 5. Tree Structure

The root or first level is drawn at the top of the figure but, of course, the reverse option is also possible. In the tree model, each state has at most one "parent" state making it a strictly hierarchical structure. If this restriction is relaxed, a network structure results. An example of a network structure is the syntax diagram in Figure 6 for a BEGIN-END block.

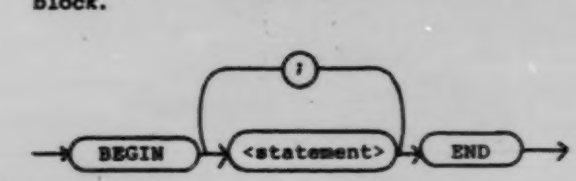


Figure 6.

Syntax Diagram for BEGIN-END Block

Another example of network structure is the model of the TERAK computer shown in Figure 7.

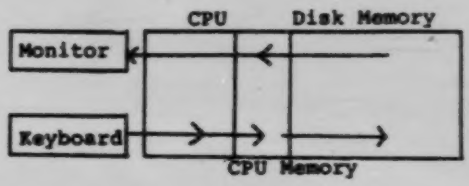


Figure 7.

Model of Flow of Information in Minicomputer TERAK

The tree and network structures consist of nodes (states) connected with directed lines. The final graphic structure to be discussed in this paper, the domain, consists of a set of areas which either overlap or are disjoint. Figure 8 illustrates a simple example of a domain structure that is a model for teaching the Boolean operations OR, NOT, and AND.

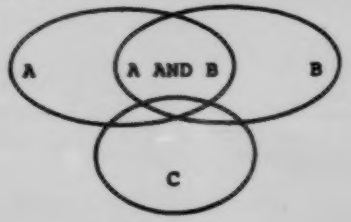


Figure 8.

Domain Structure Model for Teaching Boolean Operations

There are, of course, a wide variety of other graphic structures that might be useful in other fields of learning.

A second type of structure is the array structure. A simple one-dimensional array is often called a list, a two-dimensional array is a matrix. Array structures find their greatest utility in modeling computer memories (core, disk, etc.) and in representing data. See Figure 9 for an example of an array structure which describes the name, type, and value of PASCAL variables.

PASCAL Variables

Name	Type	Value
count	integer	43
surname	string	Jones
pi	real	3.14
alpha	character	G
condition	boolean	true

Figure 9.

Name/Type/Value Model

Mathematical structures constitute a third type of model structure consisting of a string of symbols (letters, numbers, mathematical operators, etc.). Figure 10 illustrates one type of mathematical structure, the Backus Normal Form model. It is an alternative model for the same BEGIN-END block as described by Figure 6, but with a more mathematical flavor.

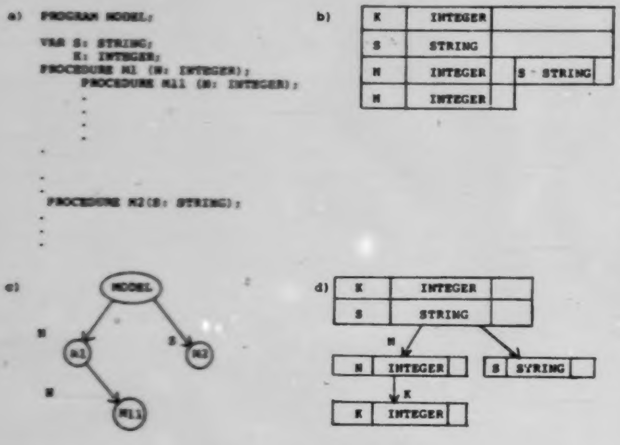
```
<block>::=BEGIN <multi-stat>  END

<multi-stat>::=<statement>|<multi-stat>;
               <statement>
```

Figure 10.

Backus Normal Form for BEGIN-END Block

For completeness, we add a fourth type, the textual structure, which is nothing but text in natural language. An article, letter, or book all contain such structures. We introduce this type of model here, not because we want to study books as textual structures, but because we want to allow textual structures as small components of larger models.

Having introduced various types of structures, we are now in a position to describe compound structures. A compound structure is one that is composed of several types of substructures For example, it is possible to combine two types of model structures (array and graphic) into one compound model as shown in Figure 11.



```
a)  PROGRAM MODEL;

    VAR S: STRING;
        K: INTEGER;
    PROCEDURE M1 (N: INTEGER);
        PROCEDURE M11 (K: INTEGER);
        .
        .
        .
        .
    .

    .
    .
     PROCEDURE M2(S: STRING);
    .
    .
    .
```

Figure 11.
Combining Array Structure (b) and Graphical Structure (c) into Compound Model (d).

The last, and probably the most significant, model structure to be described here is the parallel structure. A parallel structure is similar to the compound structure in that it consists of a collection of several more elementary models. However, a parallel structure differs from the compound structure, as the sub-models within the parallel structure remain distinct and physically separate, related only in that they represent complementary aspects or features of the same real-world object. The advantage of several parallel models over one compound model lies in the simplicity that can be retained in each complementary sub-model while at the same time representing all the necessary features of the system.

For example, the models in both Figures 5 and 7 represent certain features of the TERAK microcomputer. Figure 5 illustrates the steps necessary to have the computer perform certain actions, while Figure 7 shows the flow of information within the system. Thus, these two models form a parallel structure. Another example of parallel structure is shown in Figures 6 and 11. Figure 6 depicts the syntax of the PASCAL language (in a sense, the steps or actions in programming) while Figure 11 represents the computer memory (information flow) associated with a PASCAL program. Thus, a parallel structure consisting of one model to represent the information flow and another to represent the human performance (operator actions) seems especially appropriate and of general applicability.

IV. EVOLVING PARALLEL STRUCTURES

Effective teaching of a complex model or of a set of parallel models involves a series of steps on lessons. The models upon which these lessons are based must be carefully selected and designed so that the student's understanding grows in a smooth and orderly way. This process suggests a family of closely inter-related models culminating in the final parallel structure which has been selected to represent the system under study. The model chosen for a particular micro-lesson may be related to the models used in previous micro-lessons by:

(1) incorporating some new feature or component into a previous model.
(2) synthesis of several previous models into a new model.
(3) providing a more exact description of some part of a previous model.

As an example of this process, consider the model of a TERAK computer in Figure 7. One can introduce the concept of a work file by expanding this model as shown in Figure 12.

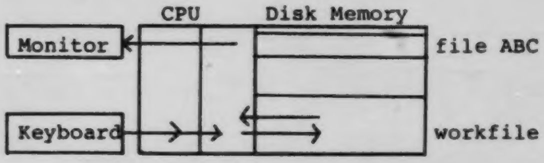


Figure 12. TERAK Information Flow

Another example of model evolution can be seen from a comparison of Figures 5 and 13. Here again a new feature is added to the model to create a more extensive model.

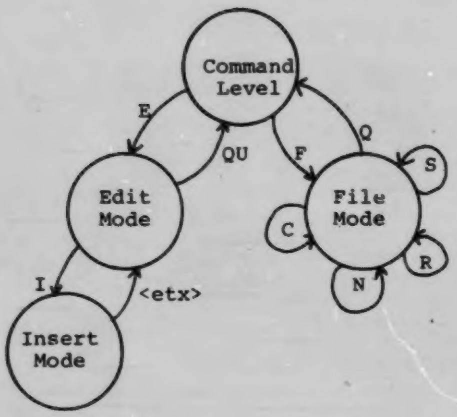


Figure 13. Tree Structure

It should be noted that the models of Figures 5, 13, 7, and 12 are inter-related and constitute an evolving parallel structure. Figure 14 depicts such a structure where M_1 and M_1' represent the models of Figures 5 and 13 and M_2 and M_2' correspond to Figures 7 and 12. Each connecting line (R_i) represents the relationship between two models or the relationship of a model to reality.

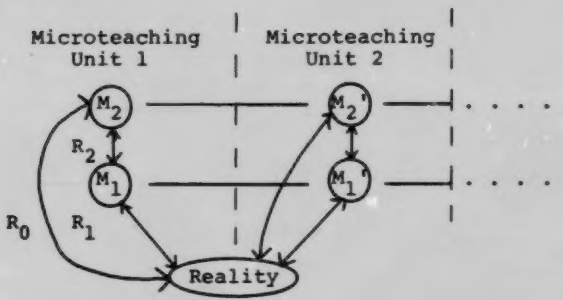


Figure 14.

Evolving Parallel Structure

V. TEACHING OBJECTIVES IN MODEL-BASED INSTRUCTION

Once an evolving parallel structure of models has been designed, it becomes fairly straightforward to define the teaching objectives. The objectives can be described in terms of the models, the relationships between the models, and relationships of the models to reality.

The purpose of each micro-teaching unit would be to solve a practical problem in some particular area. The process of problem solving using models is shown in Figure 15.

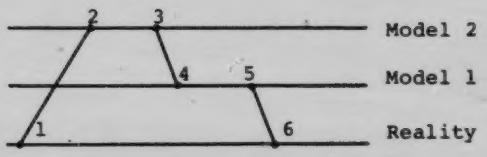


Figure 15. Problem Solving Using Models

The problem-solving process illustrated in Figure 15 consists of 6 basic points:

1. PR-Practical problem to be solved
2. Model M_2
3. MS_2- specified model M_2
4. Model M_1
5. MS_1- specified model M_1
6. PP-Practical performance

By specification, we mean the process of applying the model to a specific situation by selecting one state or path of the model.

The choice of teaching objective for each micro-teaching unit can now be specified by the triplet of transformations:

$$T_1, T_2, T_3$$

where each T is defined as follows:

$$MS_2^M = T_1(PR, M_2^M)$$

$$MS_1^M = T_2(MS_2^M, M_1^M)$$

$$PP = T_3(MS_1^M)$$

These transformations represent:

T_1 - The transformation of the Model M_2 for a given practical problem, PR.

T_2 - The transformation of the model M_1 for a given specified model, MS_2.

T_3 - Practical performance for the specified model MS_1.

Models enable us to describe teaching objectives more precisely, which is, of course, a very important condition for the success of teaching. Another advantage of this way of designing teaching objectives is connected with the analysis of problem solving, as one can concentrate on understanding and performance more than remembering.

VI. MODEL-BASED INSTRUCTION

Having prepared specific teaching objectives, we are in a position to design the micro-teaching lessons in detail. Two phases are involved: first we teach the models and the relationship (the generalization processes), and second we teach how to apply them (the specification processes). An example of this process is shown in Figure 16.

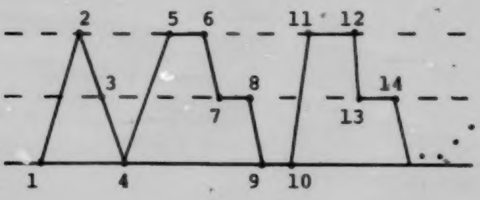


Figure 16.

Diagram of First Phase of Microteaching Lesson

A description of the steps in Figure 16 is as follows:

1. General description of a device (TERAK computer).
2. Introducing the basic elements of the Model, M_2, and relations (Keyboard, Monitor, CPU, and disk drive).
3. Basic elements of the model, M_1 (modes).
4. Description of the first task to be performed (transfer from keyboard to CPU).

5-6. Generalization of the model, M_2, so that it represents task 1 (adding information flow from keyboard to CPU).

7-8. Generalization of the model, M_1, so that it represents performance connected with task 1 (adding actions E and I).

9-15. Repetition of steps 4-9 but for task 2.

Thus, the first phase is the process of generalization of a model. It includes the introduction of new aspects to the model related to each elementary task. The second phase is the process of specification and is based on model specification as shown in Figure 15. Again, we repeat this phase several times with increasing student activity until students can solve any problem in this area.

Now we are ready to describe the basic characteristics of model-based instruction:

1. Parallel structures
2. An evolving sequence of models
3. Teaching objectives which are based on the evolving parallel structure
4. A method of teaching based on moving from a practical situation to a model and then back to practical performance.

In various practical situations, it may be appropriate to relax one or more of the above requirements for model-based instruction. In addition, the diagram of teaching (Figures 15 and 16) can be modified depending on the student's role in the creation and use of the model. When the model is an algorithmic description of a process of performance, then the process of specification of the model is impossible, and we have algorithm-based instruction rather than model-based instruction.

Some models require that certain parameters be supplied by the user to complete the model. If this is the case, then the transformations 2-3 and 4-5 of Figure 15 will consist of a series of steps instead of just one step.

Other models require some work to construct using simpler models. If the process of construction is given by some algorithmic rules, then the diagram of teaching will not be substantially changed. However, if the creation of one model is described in terms of other models, then in the process of problem solving we will have two phases in place of one. The first will be connected with model construction for the practical situation, and the second will be connected with applying this model to some practical action.

VII. THE APPLICATION OF MODEL-BASED INSTRUCTION TO COMPUTER-ASSISTED INSTRUCTION

On the basis of the principles of model-based instruction, a CAI facility has been implemented which teaches the PASCAL language on the TERAK computer. Plans are underway to also implement this facility on the TICCIT system.

It appears that MBI is a very useful method in CAI because:

1. Computer graphics provide an effective way of displaying models and their transformations. Model-based languages are much easier to implement than natural languages.
2. The strict definition of teaching objectives makes it very convenient to evaluate user responses. This is very important in tracing each step in the process of problem solving.

Three types of conversation blocks have been designed as shown in Figures 17-19.

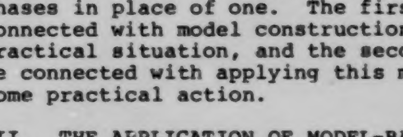


Figure 17. Simple Conversation Block

Figure 17 shows a simple conversation block. It consists of:

P → the problem to be solved
C → correction block
+ → right answer
- → wrong answer

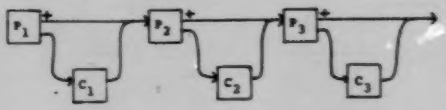


Figure 18. Teaching Conversation Block

In Figure 18, several simple conversation blocks are combined into a teaching sequence. In each simple conversation block, one of the transformations, t_1, t_2, t_3, performed by the student, is evaluated.

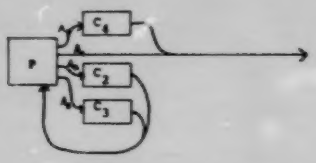


Figure 19. Evaluation Conversation Block

Figure 19, in comparison with Figure 18, shows the process of evaluating two students' solution to a problem. The students' possible responses are represented by A_1, A_2, A_3, and A_4. A_1 is the correct answer. A_2 is the wrong answer, but the answer is covered by the rules of model M_1. A_3 is the wrong answer not covered by the rules of M_1. A_4 is the exit path followed when a wrong answer is repeated. There are as usual correction blocks, C_2, C_3, and C_4, which differ depending on the response of the user.

The conversation blocks described above are, together with the information blocks, the basic elements of a CAI micro-lesson. The diagram in Figure 20 is an example of this type of CAI lesson.

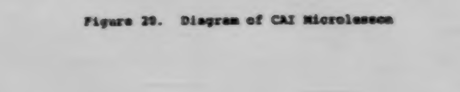


Figure 20. Diagram of CAI Microlesson

In Figure 20, IB represents an information block to be displayed on the monitor. B_1 is a simple conversation block, B_2 is a teaching conversation block, and B_3 is an evaluation conversation block. It is, of course, possible to repeat conversation blocks B_2 and B_3 until the lesson is mastered by the student.

VIII. SUMMARY

In this paper, we described the use of model-based instruction to teach the PASCAL language. In the classified set of models, we found and defined a parallel structure. We next showed how to transform this structure into a sequence of evolving models that constitute an evolving parallel structure, the basis for defining teaching objectives. The characteristics of model-based instruction were then described. Finally, the application of this teaching methodology to computer-aided instruction was shown, using as an example the teaching of the PASCAL language.

REFERENCES

1. Webster's Seventh New Collegiate Dictionary, 1965 ed. G & C Merriam Co.
2. Kolkowski, Ludwik. Nauczanie Problemowe w Szkole Zawodowej. Warszawa: WSIP, 1974.
3. Bowles, Kenneth L. Microcomputer Problem Solving Using PASCAL. New York: Springer-Verlag, 1977.
4. Jensen, Kathleen and With, Niklaus. Pascal User Manual and Report. New York: Springer-Verlag, 1970.
5. Schneider, G.M., Weiengart, S.W., and Petlman, D.M. An Introduction to Programming and Problem Solving with Pascal. New York: John Wiley, 1978.
6. Niemierko, Boleslaw. Testy Osiagniec Szkolnych. Warszawa: WSIP, 1975.
7. USCD (MINI-MICRO COMPUTER) PASCAL VERSION II.0. Institute for Information Systems, La Jolla CA, 1979.
8. Czejdo, B., Kozinski, W., Kwiatkowski, S., Kipinski, E. "Zastosowanie Maszyn Cyfrowych do Automatyzacji Przetwarzania i Przekazywania Informacji." Prace Naukowe Politechniki Warszawskiej Elektryka, 47 (December 1977).
9. Chanon, R.M. "A Course in Programming and Practice: Toward Small Systems." SIGCSE BULLETIN, 1 (February 1978): 224-228.

STRUCTURED MACHINE-LANGUAGE:
AN INTRODUCTION TO BOTH LOW- AND HIGH-LEVEL PROGRAMMING

David G. Hannay
Department of Electrical Engineering and Computer Science
Union College, Schenectady, NY 12308
(518) 370-6273

INTRODUCTION

Students in introductory computer courses are often frustrated by the unfamiliar processes involved in solving bewildering algebraic problems, particularly if their background in mathematics is weak. But these same students can learn the fundamentals of structured programming quickly and painlessly by directing the activities of a simulated robot, R1-D1.

R1-D1 is an indirect descendent of the turtle used by the LOGO group at M.I.T. to introduce elementary school children to the computer. The immediate reward of seeing a turtle move across the floor or a robot across the screen makes even non-mathematical students excited about programming. This pedagogical principle of immediate reenforcement is as valid for college students as it is for young children.

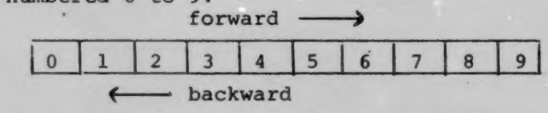
But this visible movement is one of the least significant of R1-D1's abilities, for it has the arithmetic capabilities to perform simple computations and the logic capabilities to illustrate the fundamentals of structured programming.

R1-D1 was also designed for ease of programming; each command consists of a single letter or digit. A program consists of a free-form string of commands with virtually none of the syntax rules involved in the typical programming languages; hence a student is free to concentrate on such programming techniques as decision making and loop control.

This particular robot has been used successfully in introductory programming classes by means of a simulator written in PASCAL. This simulator traces R1-D1's movements and calculations on a comand-by-command basis. Once students have been introduced to programming via the robot, the concepts of programming in a higher-level language become easier to explain, and the concepts of computer organization can be taught from a more interesting perspective.

CAPABILITIES

R1-D1 is a robot that has not outgrown his training wheels. It can move back and forth in a straight line and position itself over one of ten squares (memory cells) numbered 0 to 9:



It can be directed to move to a specific square or to take a position relative to the current square, moving forward or backward one square at a time.

R1-D1 can also do a limited amount of arithmetic. Each of the ten cells can store an integer number. It can also add or subtract from an on-board accumulator.

Decision making is accomplished by comparing the contents of the accumulator with the contents of the current cell then executing a single command (or group of commands enclosed in parentheses) based on the result of that comparison. For example, you could have the robot execute a command only if the accumulator was less than the contents of cell #5.

Finally, R1-D1 can be programmed to repeat one command (or a series of commands) a specified number of times. This structure can be enriched by the use of the decision-making commands to exit from the loop prematurely based on certain conditions. For example, it could add a number to itself seven times, or until the sum exceeded 1000, whichever comes first.

Input and output commands do not exist as such. When the simulator is run, you specify the initial contents of each of the ten cells; their contents are displayed automatically after each command is executed. However, if you wish to teach about I/O instructions as such, reading from and writing to the cells can be considered as input and output respectively. Or, while discussing computer organization, you may treat these instructions as memory fetch and store.

SUMMARY OF COMMANDS

Miscellaneous Commands:

"H"	Halt.
" "	Temporary Pause. (Allows entry of immediate mode commands.)
"N"	Make a Noise.
"@"	Execute commands from an indirect command file.

Movement Commands:

"F"	Move Forward one square.
"B"	Move Backward one square.
"0"	Move to square 0.
"1"	Move to square 1.
"2"	Move to square 2.
"3"	Move to square 3.
"4"	Move to square 4.
"5"	Move to square 5.
"6"	Move to square 6.
"7"	Move to square 7.
"8"	Move to square 8.
"9"	Move to square 9.

Data Transfer Commands:

"W"	Write contents of accumulator into current cell.
"R"	Read contents of current cell into accumulator.

Arithmetic Commands:

"Z"	Zero out accumulator.
"I"	Increment accumulator by 1.
"D"	Decrement accumulator by 1.
"A"	Add contents of current cell to accumulator.
"S"	Subtract contents of current cell from accumulator.

Skip Commands:

"E"	Execute next command only if accumulator is Equal to contents of current cell.
"G"	Execute next command only if accumulator is Greater than contents of current cell.
"L"	Execute next command only if accumulator is Less than contents of current cell.

Program Control Commands:

"P"	Perform a command as many times as the current value of the repeat count register.
"C"	Load repeat Count register from the accumulator.
"X"	Premature eXit from a perform loop.

MODES OF OPERATION

The R1-D1 can be run in any one of three different modes: immediate, direct, or programmed. In immediate mode, commands are executed as soon as the key is struck on the keyboard. This immediate feedback has proved very beneficial in eliminating some of the students' fear and has made the robot understood and enjoyed even by children in elementary school. (While in this mode the last two groups of instructions, skip and program control commands, are not allowed, as they, by definition, involve multi-character command sequences.)

In direct mode, a complete command string is given directly to the simulator which then begins execution. This allows the students to use multi-character command sequences with minimal knowledge of the host operating system. Students communicate directly with this one program without learning to use a text editor, compiler, or other system programs.

Finally, as the system text editor is introduced, students can edit and save programs for R1-D1 and modify them as they see the results generated. This approach not only helps in teaching such concepts as debugging by tracing, but also gives the students a chance to use a subset of the system text editor on a manageable piece of text since programs are typically only one line long.

APPLICATION TO VARIED AGE LEVELS

First graders enjoy watching the robot move back and forth across the screen as they enter commands in immediate mode. High schoolers can be shown graphically the relation of multiplication to addition and division to subtraction as illustrated by the sample program in the next section. College freshmen gain a glimpse of both machine language and PASCAL programming concepts.

SAMPLE PROGRAM

A program which would take the initial contents of cells 1 and 2, and put their sum in cell 3, their product in cell 4, their quotient in cell 5, and their remainder in cell 6 is given below:

```
1R2A3W
1RCZ2PA4W
Z5W1RC6WP(6R2LXS6W5RIW)H
```

This program not only illustrates several important programming concepts such as loading and storing, looping, decision making, etc., but also serves to remind students that multiplication can be performed by successive additions, and that division can be performed by successive subtractions.

The first section of the program reads the value from cell 1, adds the value in cell 2, then stores the result in cell 3.

The second section of the program picks up the first number and loads it into the repeat count register. The accumulator is cleared, then the second number is repeatedly added to it. The product is then stored in cell 4.

The third section of the program first clears the quotient to zero and sets up the dividend as the remainder. Then, each time through the "P" loop the divisor is subtracted from the dividend, and the remainder and quotient are updated. As soon as the dividend is less than the divisor, the loop exit is taken and the program halts.

One of the major disadvantages of this type of programming is the obvious lack of on-line documentation. But since the programs are so short and simple, this lack of documentation has not proven to be a handicap to students.

CONCLUSION

Decision making and loop control on R1-D1 follow the general outlines of the PASCAL IF..THEN and REPEAT statements respectively. The equivalent of a GO TO statement has been purposely omitted, thereby teaching students the concepts of structured programming from the very beginning. Parentheses can be used, when necessary, to treat a group of commands as a single command, just as BEGIN..END are used in PASCAL.

Students in introductory computer courses learn the fundamentals of structured programming quickly and painlessly by directing the activities of the robot. A few sessions with R1-D1 at the beginning of the computer science curriculum leave students both more knowledgeable and more enthusiastic about programming than they were in the pre-robot period.

SIMULATOR PROGRAM LISTING

```
PROGRAM R1D1(INPUT,OUTPUT);          { DIRECT EXECUTION VERSION }
{  THE WELL STRUCTURED ROBOT  }
{  BY;  DAVID G. HANNAY  }

VAR
     COMMAND                         {  CURRENT COMMAND  }
        :  CHAR;
     ACC,                            {  ACCUMULATOR  }
     COUNT,                          {  REPEAT COUNT REGISTER  }
     LEVEL,                          {  DEPTH OF "PERFORM" LOOPS  }
     LOC,                            {  CURRENT LOCATION OF R1D1  }
     PC,                             {  PROGRAM COUNTER  }
     PTR                             {  TEMPORARY POINTER  }
        :  INTEGER;
     CELL                            {  MEMORY CELLS  }
        :  ARRAY[0..9] OF INTEGER;
     PROG                            {  PROGRAM STORAGE  }
        :  PACKED ARRAY[1..80] OF CHAR;
     DOUBLE,                         {  SET OF PREFIX COMMANDS  }
     VALID                           {  SET OF VALID COMMANDS  }
        :  SET OF CHAR;

PROCEDURE PERFORM
     (FIRST, LAST, TIMES:  INTEGER);
     FORWARD;
```

```
PROCEDURE VALID_ASSIGN;
BEGIN
    DOUBLE := ['E','G','L','P'];
    VALID := ['H','F','B','W','R',
              'Z','I','D','A','S',
              'E','G','L',' ',
              '(',')','P','C','X']
END;  { OF VALID_ASSIGN }

PROCEDURE BLANK_SCREEN
    (ROW:  INTEGER);
BEGIN
    GOTOXY(0,ROW);
    WRITE(CHR(27),'J')
END;  { OF BLANK_SCREEN }

PROCEDURE ERROR
    (WHICH:  INTEGER);
BEGIN
    GOTOXY(0,22);
    CASE WHICH OF
      1:
        WRITELN('ILLEGAL COMMAND');
      2:
        WRITELN('OUT OF BOUNDS');
      3:
        WRITELN('PARENTHESES CHECK');
      END:  { OF CASE }
    EXIT(PERFORM)
END;  { OF ERROR }

PROCEDURE SCAN
    (VAR P1, P2:  INTEGER;
     LIMIT:  INTEGER);
    VAR P3, P4:  INTEGER;
BEGIN
    IF PROG[PC] <> '('
      THEN
        BEGIN
            P1 := 0;  P2 := 0
        END
      ELSE
        BEGIN
          PC := PC + 1;  P1 := PC;
          WHILE PROG[PC] <> ')'  DO
            BEGIN
                IF PROG[PC] = '(' THEN SCAN (P3, P4, LIMIT);
                PC := PC + 1;
                IF PC > LIMIT THEN ERROR(3)
            END;
          P2 := PC - 1
        END
END;  { OF SCAN }
PROCEDURE SKIP
    (LAST:  INTEGER);
    VAR  S1, S2:  INTEGER;
BEGIN
    PC := PC + 1;
    IF PROG[PC] IN DOUBLE THEN PC := PC + 1;
    SCAN (S1, S2, LAST);
    IF S1 <> 0 THEN PC := S2 + 1
END;  { OF SKIP }
```

```
PROCEDURE PERFORM;
    VAR
        P1, P2,
        RC:  INTEGER;
BEGIN
    RC := TIMES;
    WHILE RC >= 1  DO
      BEGIN
          GOTOXY(50,2);
          WRITE('LEVEL: ',LEVEL,'  REPEAT COUNT: ',RC);
          PC := FIRST;
          WHILE PC <= LAST  DO
            BEGIN
                COMMAND := PROG[PC];
                GOTOXY(0,12);
                WRITE(PC,': ',COMMAND,' ');
                IF (COMMAND >= '0') AND (COMMAND <= '9')
                  THEN
                    LOC := ORD(COMMAND) - ORD('0')
                  ELSE
                    IF COMMAND IN VALID THEN
                      CASE COMMAND OF
                        ' ':
                              READLN;
                        'H':
                          BEGIN
                              GOTOXY(0,22);
                              WRITELN('* * * HALT * * *');
                              EXIT(PERFORM)
                          END;
                        'F':
                          BEGIN
                              LOC := LOC + 1;
                              IF LOC > 9 THEN ERROR(2)
                          END;
                        'B':
                          BEGIN
                              LOC := LOC - 1;
                              IF LOC < 0 THEN ERROR(2)
                          END;
                        'W':
                              CELL[LOC] := ACC;
                        'R':
                              ACC := CELL[LOC];
                        'Z':
                              ACC := 0;
                        'I':
                              ACC := ACC + 1;
                        'D':
                              ACC := ACC - 1;
                        'A':
                              ACC := ACC + CELL[LOC];
                        'S':
                              ACC := ACC - CELL[LOC];
                        'E':
                              IF ACC <> CELL[LOC] THEN SKIP(LAST);
                        'G':
                              IF ACC <= CELL[LOC] THEN SKIP(LAST);
                        'L':
                              IF ACC >= CELL[LOC] THEN SKIP(LAST);
                        '(':
                          BEGIN
                              SCAN(P1,P2,LAST);
                              PERFORM(P1,P2,1);
                              PC := P2 + 1;
                          END;
                        ')':
                              ;
                        'C':
                              COUNT := ACC;
                        'P':
                          BEGIN
                              PC := PC + 1;
                              LEVEL := LEVEL + 1;
                              SCAN(P1,P2,LAST);
                              IF P2 = 0
                                THEN
                                  BEGIN
                                      P1 := PC;  P2 := PC;
                                      PERFORM(P1,P2,COUNT);
                                      PC := P2
                                  END
                                ELSE
                                  BEGIN
                                      PERFORM(P1,P2,COUNT);
                                      PC := P2 + 1
                                  END;
                              LEVEL := LEVEL - 1
                          END;
                        'X':
                          BEGIN
                              RC := 1;
                              PC := LAST;
                          END;
                        END  { OF CASE }
                      ELSE ERROR(1);
                WRITE(CHR(27),'J');
                GOTOXY(6*LOC+10,12);
                WRITELN('<',ACC,'>');
                FOR PTR := 0 TO 9  DO
                  BEGIN
                      GOTOXY(6*PTR+10,13);
                      WRITE('[',CELL[PTR],']')
                  END;
                GOTOXY(50,2);
                WRITE('LEVEL: ',LEVEL,'  REPEAT COUNT: ',RC);
                PC := PC + 1
            END;  { OF PC LOOP }
          RC := RC - 1
      END;  { OF RC LOOP }
END;  { OF PERFORM }

BEGIN {  R 1 - D 1   M A I N    P R O G R A M  }
    VALID_ASSIGN;
    WRITE('PROGRAM? '); PC := 1;
    WHILE NOT EOLN  DO
      BEGIN
          READ(COMMAND);
          PROG[PC] := COMMAND; PC := PC + 1
       END;
    COUNT := 1; LEVEL := 0; LOC := 0; ACC := 0;
    WRITELN('C E L L S');
    FOR PTR := 0 TO 9  DO
      BEGIN
          GOTOXY(6*PTR+11,10); WRITE(PTR)
      END;
    PERFORM(1,80,1);
END.  { OF PROGRAM 'R1D1' }
```

ACM Elementary and Secondary Schools Subcommittee

ACM ELEMENTARY AND SECONDARY SCHOOLS SUBCOMMITTEE PROGRESS REPORT

David Moursund*
Dept. of Computer &
Information Science
University of Oregon
Eugene, Oregon 97403
Phone: (503) 686-4408

ABSTRACT

Although the instructional use of computers at the precollege level is growing by leaps and bounds, it is still in its infancy. Over the short run many of the major problems are correctly perceived to be due to a lack of adequate or appropriate hardware, software, and courseware. But there are two other major problems that will be the dominant long-term considerations. These are the problems of securing widespread agreement upon the ultimate goal(s) for instructional use of computers and the problem of teacher training.

The Association for Computing Machinery's Elementary and Secondary Schools Subcommittee has been working for the past two years to identify some of the major problems of instructional use of computers and to help lay a foundation for progress towards their solution. Some two dozen taskgroups have been established. This paper is the introduction to a NECC/2 session in which several of the taskgroup leaders will discuss their progress.

In June 1978 the ACM Elementary and Secondary Schools Subcommittee was formed by merger of a Secondary Schools Subcommittee and a Teacher Certification Subcommittee. Since that time ES^3 has worked diligently to identify some of the major problems related to instructional use of computers at the precollege level and to help solve them. Due to very limited financial resources ES^3 in most cases can only hope to provide some leadership and to lay a foundation which may help lead to long term solutions.

This same two-year time span has witnessed a massive influx of computer facility into the schools. Reliable data on how much hardware has become available has not been collected. We know, however, that total sales of microcomputers by Apple, Commodore, Radio Shack, and a number of other companies total in the hundreds of thousands. The state of Minnesota has long been recognized for its leading role in the instructional use of computers. Through the Minnesota Educational Computing Consortium it has made time-shared computing available to almost every school in the state's precollege public education system. During the past two years these same schools have added approximately 1,000 microcomputers, while continuing to increase their use of the time-shared system. Minnesota has a population of about 4 million, or a little less than 2% of the total US population.

---

*Dave Moursund is Chairman of the ACM Elementary and Secondary Schools Subcommittee. He is also editor of The Computing Teacher, a professional journal aimed mainly at elementary and secondary school teachers interested in instructional use of computers.

One can get additional insight into this massive growth of microcomputer use in education by talking to a variety of teachers and by attending local and national professional meetings, as it has been my privilege to do. For example, each year the northern California affiliate of the National Council of Teachers of Mathematics holds a meeting at Asilomar, near Monterey, California. Usual attendance is about 1,500 educators. Bob Albrecht, well-known author and leader in the computer education field, has attended this meeting for many years. According to Bob the 1978 meeting included a modest amount of computer activity, and there were 10 microcomputers available for demonstrations and hands-on use.

At the most recent 1979 meeting there were computer-related talks scheduled in parallel with every math session. There were some 66 microcomputers available for hands-on experience, and there was a well-organized software swap. The software swap was set up by the Computer-Using Educators (CUE), a northern California group that is less than two years old. By the end of the Asilomar meeting CUE had well over 200 members. (1)

At the Asilomar meeting I talked with dozens of teachers who are just getting started in the instructional use of computers. The most typical question went approximately like this: "Our school has $X to spend for computers. What should we do?" The typical figure mentioned ranged from a thousand dollars or so up to many times that amount.

Two conclusions seem evident. First, many people with very little formal training or experience in the instructional use of computers are now becoming involved; indeed, they are being asked to make major decisions that will affect how computers are used in their schools. Second, while the primary concern is still hardware, there is growing awareness on the part of novice users that software is also a major issue.

On the average, the novice computer educators know little about work done by others nor of the higher level problems faced by the field of instructional use of computers.

WHERE ARE WE HEADED?

In my opinion one of the major problems of instructional use of computers is there is little agreement as to where we are headed. The overall long-term goals are neither clearly understood nor widely accepted by the people who will be involved in implementing them.

Over the past decade computer literacy for all students has emerged as the major goal in instructional use of computers at the precollege level. Initially computer literacy tended to mean computer awareness. It was thought by many to be adequate if students could come to understand some of the capabilities and limitations of computers and thus gain some insight into how computers were affecting the world and their lives. There was relatively little mention of giving students substantial training in use of computers or integrating their use into the curriculum. Initially, at least, it was clear that resources to do so were not available within the foreseeable future.

But the future proved difficult to foresee accurately, and large-scale integrated circuitry became commonplace. As the price of computers dropped and availability increased, the meaning of computer literacy changed. Now the expression tends to mean a functional, working knowledge of computers.

The analogy with reading and writing literacy expressed by Art Luehrmann (2) is highly instructive. We can imagine being involved in education at the dawn of the invention of reading and writing and entering into discussion as to their role in education. We can imagine educators getting bogged down on issues such as the brand of pencils and paper that are best or lamenting the poor quality and small quantity of books that are available. But these turn out to be short-term issues, and are certainly not the major long-term problem. Over a time span of a few thousand years, and aided by the invention of movable type and high-speed printing presses, reading and writing come to be integrated into every aspect of human intellectual activity.

What have proved to be continuing problems in reading and writing are illustrated by looking at examples, such as in mathematics and music. In each field there has been a need to develop appropriate symbols, notations, and vocabulary in order to represent the key ideas. Key ideas had to be developed and preserved. As this knowledge accumulated, educators are faced with the problem of how to teach it and what students should learn.

Now shift your attention back to the problem of instructional use of computers. If our technologically oriented society continues to prosper then we can easily imagine that eventually computers will be as readily and conveniently available as books, pencils, and paper are today. Within that framework a single clear-cut goal for instructional use of computers seems evident to me. We want to integrate use of computers into every aspect of human intellectual activity in the same manner that we have done for reading and writing. The computer is a universal tool, a new aid to problem solving, and we want all students to develop a high level of functional knowledge and skill in its use.

This ultimate goal will take many decades, centuries, or even millenia to accomplish. Progress will be made by identifying problems that face current educators and working to solve these problems. The ACM ES^3 has identified about two dozen problems and established a like number of task groups to work on them. The next several sections of this paper discuss some of these problems.

SHOULD COMPUTERS BE USED IN EDUCATION?

There is growing support from teachers, school administrators, school board members, parents, and others for the instructional use of computers in education. But much of this support is still tentative, and much of it is based upon incorrect insights as to how computers will affect education. Robert Taylor of Columbia Teachers College heads the ES^3 Arguments taskgroup. His review of the literature led him to assemble a book of readings (3) on the arguments in favor of instructional use of computers that have been so well expressed by various people over the past 15 years. Perhaps the main conclusion to draw from Taylor's work is that many people are busy reinventing the wheel. Most people currently entering the computers in education field are not taking the time to learn what is already known and thus are expending considerable effort redoing what has already been done.

HARDWARE, SOFTWARE, AND COURSEWARE PROBLEMS

Several of the ES^3 taskgroups are concerned with the problems of hardware, software, and courseware; as mentioned earlier, many people view them as the major issues that need to be resolved. The hardware and software problems are closely related. We have a growing number of options in educational hardware, with the quality and capability of these machines continuing to improve. But there is a lack of software compatibility between different manufacturer's machines, and we are just beginning to see how massively difficult the software problem really is.

We can better understand the software/courseware problem by examining one of the major directions of expanded computer activity, computer-assisted instruction. The idea is for the computer to take over some of a teacher's functions, interacting with students to enhance student learning. Material is needed at every grade level and in every discipline, and the past 20 years of experience of the PLATO project, as well as many other computer-assisted instruction projects, points out the difficulty of developing good quality instructional software and courseware. Developing good software and computer-oriented courseware is more difficult than developing the more traditional materials currently in use. Moreover, the current market is small, the number of people involved in developing materials is modest, and the machinery for coordinated national efforts in development or distribution is not yet well established.

It is clear, then, that regardless of hardware progress, software and courseware will continue to be major problems for many years to come. Some federal funding is being made available, however. For example, Judy Edwards (4) heads a three year, $200,000 per year project to work on educational software for microcomputers.

Another approach is being strongly encouraged by ES^3 and has also received the backing of the International Council for Computers in Education (5). Local or statewide groups of computer-using educators who can readily participate in software and courseware exchange are being formed. Such local groups bring people together for personal interaction and are proving to be a highly effective mode of information dissemination. Those interested in starting such a group please contact the author of this paper.

CURRICULUM CONTENT QUESTIONS

Computers are a general aid to problem solving in every academic discipline, although they currently are much more used in some areas than in others. ES^3 has taskgroups concerned with computer uses in mathematics, the sciences, the social sciences, the humanities (including art and music), business, and vocational education. The task in each case is fairly similar. Eventually use of a computer as an aid to knowing the subject matter and to solving problems from these disciplines will be routine. So far, however, relatively little progress has occurred.

We find the most progress in mathematics. Calculator use in secondary school mathematics is now common, especially in the more advanced courses; strangely (in the author's opinion) use of calculators in the remedial math courses has been more slow to gain acceptance. The use of computers is taught in some math courses, but we seldom find a math course where the math content has changed to reflect what computers can do or how one uses them in problem solving. Almost always the computer is an add-on topic and is used in conjunction with learning the traditional topics in the traditional manner.

In very few secondary schools one can find a significant change in the science or business curriculum due to computers. As in math, computer use tends to be add-on in nature, to help students learn the traditional content. Thus it is a very rare student who emerges from high school with a functional knowledge of the computer as an aid to problem solving in a variety of disciplines.

The problem here is immense. The content and coursework for every discipline needs to be rethought and redone in the light of computers and their capabilities.

ELEMENTARY SCHOOL

A single taskgroup is attempting to bring order out of chaos in elementary school education, which represents about half of all of precollege education. Although this group is not expending much energy on the issue of calculators, that particular issue gives good insight into some of the overall difficulties faced by the field of instructional use of computers.

Calculators are now quite inexpensive and reliable. Even at the retail level one can buy a 4-function machine with 4-key memory and liquid crystal display for under $10. Since such a machine will last for years, an elementary school could provide its students with essentially unlimited access to calculators at a cost of perhaps $2 per student per year. The use of calculators at this level has been studied in many research projects and has received the backing of such organizations as the National Council of Teachers of Mathematics and the National Council of Supervisors of Mathematics. Many books of calculator materials are now available for use in the elementary school.

But calculators have had essentially no impact upon the elementary school mathematics curriculum! Speculations as to why would fill at least an entire paper; however, it seems that a major factor is that teachers lack appropriate knowledge and skills.

What, then, can we expect to happen with computers in the elementary school? One answer is computer-assisted instruction, with the main emphasis being on the computer taking over some of the instructional processes currently handled in other ways. Proponents talk about teacher-proof materials and point out inadequacies in the current instructional program. Opponents point out the inadequacies of computerized instructional materials and discuss the wisdom of this type of change in our instructional delivery system.

Very few elementary schools currently attempt to teach students about computers. We have only modest insight as to what is appropriate. Again, the major drawback is teacher knowledge.

TEACHER KNOWLEDGE

Every teacher knows how to read and write and makes use of this knowledge in teaching. Every teacher has substantial insight into the use of reading and writing as a tool to learning the disciplines s/he teaches, and in solving the problems of these disciplines. The great bulk of instructional materials builds upon students' abilities to read and write

Contrast this with teacher knowledge of computers! It is a truly rare school that has even one teacher who uses computers as an everyday tool in coping with the problems of his/her discipline. This, then, is the major problem. We are asking computer illiterate teachers to help students to become computer literate at a functional level.

This problem is being attacked in many ways. Among these are teacher certification requirements, pre-service courses, in-service courses, self-directed study, and so on. Many teacher organizations recognize the problem and are including computer talks and computer tutorials in their professional meetings. All of these things are necessary, and all are helping. But progress seems slow relative to the magnitude of the task. The educational world has yet to accept the use of computers in precollege education as a major goal, and thus to begin to devote the resources necessary for rapid progress.

REFERENCES

1. Computer-Using Educators. Founder and president of this organization is Dr. William (Sandy) Wagner, Mountain View High School, Mountain View, CA 94041.
2. Leuhrmann, Arthur. "Should the Computer Teach the Student, or Vice-versa?" Proceedings of the Spring Joint Conference, 1972. Reprinted in Creative Computing, vol. 2, no. 6, Nov-Dec 1976.
3. Taylor, Robert. Tutor, Tool, Tutee: The Computer in the School. Teachers College Press, scheduled for publication in 1980.
4. Edwards, Judy. She is Director of Computer Technology Northwest Regional Educational Lab, 710 S.W. 2nd, Portland, OR 97204.
5. International Council for Computers in Education. This non-profit professional organization publishes The Computing Teacher, a journal for educators. Seven issues are planned for academic year 1980-81, and the price for US subscribers is $10. Write to ICCE, %Computing Center, Eastern Oregon College, La Grande, OR 97850.

COMPUTING COMPETENCIES FOR SCHOOL TEACHERS

Robert P. Taylor
Teachers College
Columbia University
(212) 678-3484

James L. Poirot
North Texas University
(817) 788-2521

James D. Powell
North Carolina State University
(919) 737-2858

INTRODUCTION

On December 4-5, 1978, the Elementary and Secondary Schools Subcommittee (ES3) of the ACM Curriculum Committee met in Washington, DC to begin formally laying out curricular and teacher training guidelines for the integration of computing into the elementary and secondary schools of the country. Building upon a lengthy initial discussion and a review of earlier papers and documents dealing with the same problem, the subcommittee outlined what it would attempt to do. This process identified several tasks that together would constitute the overall work of the subcommittee. Finally, on the basis of interest and expertise, each participant in the ES3 meeting was assigned to a working task group to further define and carry out one of the identified tasks.

Among the tasks identified was one dealing with teacher training: to define the scope and substance of teacher training needed to integrate computing into the schools. Accordingly, a teacher training task group was formed.

This paper is a product of that task group. It deals only with a subset of the issues and areas related to designing overall, computer-literate teacher training. To appreciate its focus and accept some of its omissions, one should be aware of the following constraints that task group placed on itself. First, it was unanimously agreed that definitions should be in terms of competencies to be achieved rather than in terms of programs or courses to transmit those competencies. Second, because the computing competencies needed by the teacher who must teach computing as a subject are more extensive than those needed by other teachers, the competencies needed only by the computing teacher should be treated as a separate module. Third, though integrally related to each other, the competencies needed by the teacher are quite different from those needed by the teacher's teachers, the staffs of institutions actually doing the teacher training. It was agreed, therefore, that specifying the competencies needed by the teacher's teacher would be a separate module of work. It was also agreed that it should only be undertaken after the competencies needed by the teacher had been specified.

The task group saw the competencies needed by teachers at the school level as belonging to one of three sets. The first set encompasses those basic universal computing competencies required for any school teaching, regardless of level or subject. The second encompasses those additional computing competencies needed only by the teacher who must teach computing as a subject in its own right. The third encompasses additional computing-related, subject-specific competencies needed by teachers of subjects other than computing.

This paper outlines the competencies in all three sets. It incorporates critical suggestions received as a result of wide circulation of two earlier papers on the topic (1,2). We also trust it will stimulate useful discussion and criticism. We hope it provides some guidance to those wondering what teachers should know about computing.

COMPUTING COMPETENCIES NEEDED BY TEACHERS

Three sets of computing competencies follow. The first (1.0) includes those which all teachers must have, regardless of their level or discipline, even if that discipline is the teaching of computing itself. The second set (2.0) includes those needed only by the teacher of computing as a subject. It should be noted this second set presupposes the first. The third set (3.0) includes additional competencies for teachers who use computing to support or enhance instruction in subjects other than computing. Every teacher should acquire the competencies listed in the first set (1.0) and the competencies listed in either the second set (2.0) or the third set (3.0).

1.0 : Universal computing competencies needed by all teachers

These are computing competencies which all school teachers should have to teach effectively in a society permeated by computers. They relate to either or both of two goals: (1) to understand computing and (2) to use computing. They can be stated partially in terms of competencies listed in ACM's "Curriculum '78" (3) and partially in terms of different competencies derived from other sources. A number of such other sources are listed in the references at the end of this paper. They reflect the abundance of diverse work that has taken place in the past decade relating computing to education.

1.1 : Competencies

In terms of these universal competencies, every teacher should:

C1.1 be able to read and write simple programs that work correctly and to understand how programs and subprograms fit together into systems;

C1.2 have experience using educational application software and documentation;

C1.3 have a working knowledge of computer terminology, particularly as it relates to hardware;

C1.4 know by example, particularly in using computers in education, some types of problems that are and some general types of problems that are not currently amenable to computer solution;

C1.5 be able to identify and use current information on computing as it relates to education;

C1.6 be able to discuss at the level of an intelligent layperson some of the history of computing, particularly as it relates to education; and

C1.7 be able to discuss moral or human-impact issues of computing as they relate to societal use of computers generally and educational use particularly.

The above competencies should be transmitted within the general preparation programs for all teachers by having those programs include the topics listed below (T1.1 through T1.5). For those being trained to teach computer science, those topics will represent only a small subset of what must be learned about computing and its uses (see Section 2.0). For all other teachers, however (apart from the subject-specific competencies covered in Section 3.0), these topics cover much of what must be learned about computing by the teacher who is to be minimally literate.

1.2 : Topics of Study

T1.1 **Programming Topics:** Includes development of simple algorithms and their implementation in a programming language, programming style, debugging and verification, and task-specific programming for educational applications.

T1.2 **Computer Terminology:** Includes software (e.g., operating systems, time sharing systems) hardware (e.g., CRT, tape, disk, microcomputers) and miscellaneous items (e.g., documentation, testing, vendors).

T1.3 **Classic Applications of Computing in Education:** Includes representative experience with problem solving and text manipulation; simulation, drill and practice, and complex tutorial systems including complete student progress record keeping; and educational administrative systems.

T1.4 **Human/Machine Relationships:** Includes artificial intelligence, robotics, computer assistance in decision making (e.g., medical, legal, business), simulations, and computers in fiction.

T1.5 **Information on Computers in Education:** Includes periodicals, important books; on-line inquiry sources such as ERIC; professional societies such as ACM, AEDS, NCTM; time-sharing networks, networks of compuuter users; and hardware vendor groups.

In T1.1 procedures or algorithms are at the heart of computing so teachers should learn what they are by implementing simple examples such as: a procedure to average a class's grades, a game to guess what number the computer is thinking of, or a procedure to display a large box on the screen and then make it shrink to disappearance.

Teachers need to be able to implement such procedures in only one language but should be able to read at the same, or a greater level of difficulty in a second so that the idea of a language, its strength and limitations, springs from personally experiencing functional differences between two languages. Within the limitations of simple programs,teachers should be taught to write well-structured code, easily readable by others, and to document their code in acceptable fashion.

T1.2 should be integrated throughout the course of study. In order to successfully use computing the vocabulary of the field must be understood. What is the difference between a tape, a floppy disk, and a disk? Why use one over the other? These are general ideas but some minimal understanding of them is necessary to use computers.

T1.3 should certainly familiarize the teacher with several of the existing well-developed CAI systems cited in Section 3.0 below. Teachers will not develop new ideas about what could be done in their areas unless they see the best of what has been done; neither will they get a full understanding of what can not be reasonably well done by computer without such exposure. For example, acquaintance with the physics system designed by Bork (4), with CCC drill and practice systems based on Suppes's work (5), or with PLATO work (6), should serve to acquaint the teacher with the CAI issues.

Since many teachers end up in administration and since administrative uses of computers affect the teacher, some introduction to one or more representative administrative systems should be included. A student record system would probably be a reasonable choice for illustrative study; it deals with information familiar to the teacher without using financial details some might find difficult to understand.

Teachers should also be familiar with super calculator modes of computer use as a classic application. As home computers become more common, perhaps little formal work in this area will be necessary. Clearly,word processing must be covered; every teacher does so much word processing manually that none should be left ignorant of how much word processing help the computer can give.

With respect to T1.4, the long range and the immediate implications of computing as a form of artifical intelligence should be taught. The excitement of learning to think about thinking and of contemplating the powers and limits of human intelligence are so significantly linked with computing experience that this aspect must be studied; the opportunity is too great to pass up. Artificial intelligence may best convey both the power and limitation of computing in education. Acquaintance with any of several perspectives on this experience is essential and can be taught using such projects as the LOGO work (7) or the SOLO work (8). A growing body of fiction about computing can also contribute effectively to the teacher's insight into the emerging world (9,10).

With respect to T1.5, teachers must know where to look to keep abreast in this rapidly changing field. Course work and instruction should therefore routinely call attention to and require that the trainee use a range of sources about computing and education.

The ideas presented above represent a minimal set of competencies which every teacher should obtain. The topics presented provide a framework to achieve this minimal level of competency. In addition to these competencies, every teacher should also acquire the competencies listed in either Section 2.0 or Section 3.0.

2.0 : Competencies needed for the teacher of computing

While every classroom teacher should have the general set of computing competencies suggested above, the teacher of computing needs more. The likelihood that he or she will, in addition to teaching, be forced to function as a general resource to faculty, administration, and students only increases the need for more extensive competency in computing.

Since much of the knowledge required for such a teacher is similar to that required of anyone desiring to be a computer professional, many of the computer competencies defined in the recent ACM Curriculum Committee report, "Curriculum '78" (3), apply to the teacher as well. This sector therefore relies extensively upon that report.

2.1 : Competencies

The core material recommended for teachers of computer science represents essential elementary material, as well as material especially designed for educators. Computer science teachers should:

C2.1 be able to write and document readable, well-structured programs and linked systems of two or more programs;

C2.2 be able to determine whether they have written a reasonably efficient and well-organized program;

C2.3 understand basic computer architectures;

C2.4 understand the range of computing topics that are suitable to be taught, as well as the justification for teaching these topics;

C2.5 know what educational tools can be uniquely employed in computer science education.

The first three competencies are of the sort commonly needed by all computing professionals, and are listed in "Curriculum '78" as among those to be covered by the undergraduate computer science degree program. Competencies C2.4 and C2.5 are not commonly needed by all computing professionals. They are essential only in the preparation of computer science teachers.

Formal training in mathematics is considered crucial for the teachers of computer science. While the specification of this mathematical content is considered beyond the scope of this report, it should be noted that this content may include topics which are frequently not part of formal mathematics programs (e.g. finite mathematics, statistics).

For individuals who are to serve as a computer resource person for their school or school system, two additional competencies have been identified.

C2.6 Develop the ability to assist in the selection, acquisition, and use of computers, interactive terminals, and computer services suitable to enhance instruction.

C2.7 Be able to assist teachers in evaluation, selection, and/or development of appropriate instructional materials that use computing facilities.

2.2 : Topics of Study

These competencies should be gained through a series of courses and other vehicles developed through joint efforts of teacher education programs and the computer science program. We present below a list of topics that should be included in the program.

T2.1 Programming Topics: Includes advanced algorithms, programming languages, blocks and procedures, programming style, documentation debugging and verification, elementary algorithm analysis, applications.

T2.2 Software Organization: Includes computer structure and machine language, data representation, symbolic coding and assembly systems, addressing techniques, macros, program segmentation and linkage, linkers and loaders, systems and utility programs.

T2.3 Hardware Organization: Includes computer systems organization, logic design, data representation and transfer, digital arithmetic, digital storage and accessing control, I/O, reliability.

T2.4 Data Structures and Filing Processing: Includes data structures, sorting and searching, trees, file terminology, sequential access, random access, file I/O.

T2.5 Computers in Society: Computers and their effects on governments, careers, thought, law, personal behavior; privacy and its protection; information security and its preservation.

T2.6 Teaching Computer Science: Includes (1) knowledge of learning theories as they apply to learning about computers, (2) knowledge of *several* appropriate curricular scope and sequences for a variety of program goals (e.g., literacy, careers, college preparation, personal problem solving).

"Curriculum '78,"[3] along with a vast amount of research in computer education, supports the inclusion of topics T2.1 through T2.5. Knowledge of programming topics, software organization, etc., are essential for the computer professional of today.

The teaching of computing is a unique computer profession. Knowing how to program, however, does not, in itself, qualify a teacher for teaching computer science. Materials on why and how to teach computer topics included in T2.6 are invaluable to the teacher of computing and should be included within a program of study training such teachers.

Competencies C2.6 and C2.7 of the previous section are required for those serving as computer resource personnel for a school system. These competencies should be gained through the computer science program and the teacher preparation programs. The following topics will assist in developing the required competencies.

T2.7 Advanced Topics in Computer Science: includes advanced topics in computer organization, operating systems, architecture; database systems; computer communications.

T2.8 Computers in Education: includes detailed knowledge of learning/teaching research as it has implications for effective design of institutional computing systems and administrative uses of computing in our educational setting.

Including study of computers in education will increase the teacher's ability to assume a role of leadership in providing direction to a school system in integrating computing into its curriculum. This additional computer background should allow the computing teacher to act as a resource person to assist in fostering development and implementation of computing throughout the school, even when the other teachers know nothing of computing.

3.0 : Subject-specific computing competencies needed by teachers

In addition to the set of universal competencies needed by all school teachers, there are additional level- and subject-specific competencies which teachers should have. Any teacher will require at least one of these, but no one of them will be universally appropriate for all teachers. The definitions of those competencies spring entirely from the vast and highly diverse body of experience with using computing in education over the last decade. Sources representing some of this work are listed in the bibliography. The competencies can be stated generally, irrespective of the teacher's eventual level of subject; the topics, though, will vary considerably, depending on both.

3.1 : Competencies

In terms of these subject-specific competencies, the teacher should:

C3.1 be able to use and evaulate the general capabilities of the computer as a tool for pursuing various discipline- or level-specific educational tasks;

C3.2 be able to use and evaluate alternative hardware and software systems designed to function as tutors or teacher aids;

C3.3 be familiar with information and quantitative techniques of study in the (teacher's) subject.

These competencies should be developed by the teacher preparation program, tailored to suit the trainee's intended teaching level and subject. We will not present an exhaustive list of topics corresponding to these subject-specific and level-specific competencies. Instead, we will present model topics for a few selected subjects and levels.

3.1.1 : Topics of Study for Teachers of Early Childhood (Primary Grades 1-4) -- TEC

TEC3.1 Computerless preparation for computing: Experience with a wide range of computerless but computing-related activities that children can participate in to enhance their readiness to understand and work with computers.

TEC3.2 Games and Simulations: Experience with a wide range of games and simulations that stimulate children to explore and better understand fundamental concepts and strategies of learning.

TEC3.3 Tutorial systems: Experience with simple and complex tutorial systems focusing upon mathematics, spelling, reading, and other elementary topics, including bi-lingual variations of such systems.

TEC3.4 Exploratory programming systems: Experience with well-developed exploratory systems where child-appropriate I/O subsystems such as robots are programatically manipulated by the child in a discovery or problem-solving approach.

Less work has taken place in the area covered by TEC3.1 than one might expect. Despite the likely wide-spread availability of microprocessors in the immediate future, computerless computing activities can still be very useful. By contrast with heavily machine-dependent activities, they provide a more contemplative, less involved opportunity to examine some of the fundamental ideas connected with computing. They thus allow those using them to deepen their understanding even if they have access to computers. Typical examples may be found in some of Papert's work (11) and in Taylor (12).

Vast quantities of games and simulations are available for TEC3.2, but careful choices should be made in selecting them. Many are not well-written, either from a programming point of view or from a child-user point of view, and no game or simulation should be selected unless it succeeds in both areas. Some of the best work in this area at this age level came out of the People's Computer Company under the initial stimulus of Albrecht (13).

With respect to TEC3.3, though, many tutorial systems have been developed, not all of them are good. The experience of the teacher should certainly include at least one good system and some discussion of what lies behind it. The work of the CCC group under Suppes is certainly a worthy example in this category (14).

Finally, new exploratory systems relevant for TEC3.4 are appearing, but the pioneering work is still only for illustration. In particular, the XEROX work which produced SMALLTALK (15) and the LOGO work, particularly as Seymour Papert has advertised and sustained it (16), is outstanding.

Microprocessors can be the primary machine used, but not so much that the trainee is left ignorant of the advantages of larger systems.

3.2.2 : Topics of Study for Teachers of Foreign Language -- TFL

TFL3.1 Games and Simulations: Experience with a wide range of games and simulations designed to provide cultural background and informal language learning, using the culture as context and the language as the medium of communication in the game or simulation.

TFL3.2 Tutorial Systems: Experience with tutorial systems designed to enhance the learning of a foreign language through a carefully arranged body of interactive experiences driven by competency-based, computer-administered testing.

TFL3.3 Foreign Language Text Editing: Experience with a powerful text editor used to create and manipulate texts in a foreign language.

Under TFL3.1, simulations based on relevant activities and situations in the language-culture can provide insight into the language difficult to obtain in other ways. These, and many popular computer games, should also be used to provide a more informal language practice for learners. This practice can take two forms: (1) translating and (2) informally using the language. Teachers should practice translating all user text of appropriate games and simulations into the target language, thus preparing to have their students do such translation. Teachers should also have wide experience with playing games whose text is entirely in the target foreign language and which expect all player responses in that language. Such playing in a foreign language can be a valuable informal enhancement. Experience with a wide range of such games and simulations may also suggest new, more appropriate ones which the teacher can create or have others, including students, create. Some attempt to create such new material (or new variations of old material) should be part of every foreign language teacher's training. Naturally, where audio is available, it should be appropriately used.

There are many examples of language drill and practice suitable for use under TFL3.2. Work such as that done by Suppes (17) at Stanford should certainly be familiar to language teachers, though alternatives certainly exist (18). Work in this area should rely as heavily upon audio and graphics as possible, thus cutting down on the automatic tendency to always translate from the native language and to develop competence only in reading and writing the foreign language.

TFL3.3 should ensure that teachers use a suitable computer text editor to manipulate text in the target foreign language. With appropriate accent mark capabilities, such editors can encourage language learners to practice much more prose writing and thus enhance their overall command of the language.

3.2.2 : Topics of Study for Teachers of Physical Science -- TPS

TPS3.1 Exploratory Programming Systems: Experience with well-defined exploratory systems through structured, discipline-appropriate languages. Systems must include graphic capabilities, be programmably controllable by the student, and be oriented to discovery through problem-solving activities relevant to the physical sciences.

TPS3.2 Tutorial Systems: Experience with tutorial systems designed to enhance learning of the physical sciences through a carefully arranged body of interactive experiences driven by competency-based, computer-administered testing.

TPS3.3 Games and Simulations Experience with stand-alone games and simulations designed to enhance understanding of specific physical phenomena or significant past experiments.

TPS3.4 Classroom/Laboratory Management: Experience with automated management of people, learning, time, and resources including automated inventorying, laboratory information/reference systems; in general, uses of the computer to provide the science teacher with more time to work with individuals.

TPS3.5 Data Collection and Analysis: Experience with systems which collect and analyze data including on-line gathering and analysis of experimental data and process control systems which collect, analyze and provide feedback to the system.

The possibilities under all three topics for physical science teachers have been extensively explored already by Bork (19) and others (20). Their careful work and well-documented analysis should be extended and incorporated in the training of physical science teachers. Such training should include experience with relevant examples which illustrate the three topics; it should also require each trainee to construct selected, similar, small modules of computer-supported instruction as a normal part of teacher training.

SUMMARY

This paper has addressed the computing competencies needed by pre-college teachers. These competencies are listed in three groupings based on the involvement of the teacher in computer-related activities. Because of the variation of subject matter, implementation examples have been given for teachers of grades 1-4 (early childhood), for teachers of foreign languages and for teachers of physical science. These examples are not meant to be all inclusive but to indicate the level of necessary background knowledge.

Before graduating from a teacher training program, all teachers should be required to acquire the first set of competencies and either the second or third set. This requirement will prepare each teacher to use computing in the classroom.

No attempt has been made to package the competencies into specific courses. It is felt that each environment will possibly require a different technique for the introduction of the material.

Another group of individuals that need to be considered are the current in-service teachers. The competencies described above are as important for them as for our future teachers. In-service courses must be developed to provide the indicated background for in-service teachers.

List of Contributors

Vivian Coun, University of Missouri-Rolla
Richard Dinnis, University of Illinois
Dan Isaacsen, University of Oregon
David Moursund, University of Oregon
John W. Hamblen, University of Missouri-Rolla
Jerome R. Kaczorowski, Bremen High School, Midlothian, Illinois
James Lockard, Buena Vista College
Dick Ricketts, Multnomah County Education Service District, Portland, Oregan
Stan Troitman, Wheelock College

Cited References

1) Poriot, James, James Powell, John Hamblen, Robert Taylor. "Computing Competencies for School Teachers — A Preliminary Projection for the Teacher of Computing." National Educational Computing Conference Proceedings, Iowa City, 1979.
2) Taylor, R.P., John Hamblin, James L. Poriot, and James D. Powell. "Computing Competencies for Teachers -- A Preliminary Projection for All but the Teacher of Computing." National Educational Computing Conference Proceedings, Iowa City, 1979.
3) Curriculum Committee on Computer Science (C3S). "Curriculum 78: Recommendations for the Undergraduate Program in Computer Science." (Preliminary Draft, dated Autumn 1978).
4) Bork, Alfred. "Learning to Teach Via Teaching the Computer to Teach." Journal of Computer-Based Instruction, November 1975, vol 2.
5) Suppes, Patrick. "Computer-Assisted Instruction at Stanford" in Man and Computer, Karger, 1970.
6) Smith, Stanley and Bruce Arne Sherwood. "Educational Uses of the PLATO Computer System." SCIENCE, April 1976 , vol 192.
7) Papert, Seymour. "Teaching Children Thinking," Logo Memo 2, MIT AI LAB, October 1971.
8) Dwyer, Thomas. "The Art of Education: Blueprint for a Renaissance" Creative Computing, Sept/Oct 1976.
9) Mowshowitz, Abbe (editor). Inside Information: Computers in Fiction. Reading, Mass.: Addison-Wesley, 1978.
10) Taylor, Robert P. (editor). Tales of the Marvelous Machine: Thirty-Five Stories of Computing. Morristown, N.J.: Creative Computing Press, 1980.
11) Papert, Seymour. "Teaching Children to Be Mathematicians VS. Teaching About Mathematics." Logo memo 4, MIT AI LAB, July 1971.
12) Taylor, R. P. "Computerless Computing for Young Children or What to Do till the Computer Comes" in Proceeding of the 2nd World Conference on Computers in Education. North Holland/Amsterdam, 1975.

13) PCC (collective authorship). What to Do after You Hit Return. Menlo Park, Calif.: People's Computer Company, 1975.
14) Suppes, Patrick and Elizabeth Macken. "Evaluation Studies of CCC Elementary School Curriculums 1971-1975." Computer Curriculum Corporation, Palo Alto, 1976.
15) Kay, Alan. "A Personal Computer for Children of All Ages" in Proceedings of ACM National Conference, Boston, August 1972.
16) Papert, Seymour. "Personal Computing and the Impact on Education." Edited transcription of a talk delivered at the Gerald P. Weeg Memorial Conference, as printed in the proceedings, Computing in College and University: 1978 and Beyond, University of Iowa, 1978.
17) Suppes, Patrick, Robert Smith, Marian Beard. "University-Level Computer-Assisted Instruction at Stanford: 1975." Instructional Science, 6 (1977), Elsevier Scientific Publishing Company, Amsterdam.
18) Tanabe,Y. et al. "CAI in Language Education" in Proceedings of 2nd World Conference on Computers in Education, North Holland/Elsevier, 1975.
19) Bork, Alfred. "Preparing Student-Computer Dialogs -- Advice to Teachers." PCDP, U. C. Irvine, July 1976.
20) Dwyer, Thomas A. "Some Principles for the Human Use of Computers in Education." International Journal of Man-Machine Studies 3, July 1971.

Other References

21) Atchinson, William F. "Computer Science Preparation for Secondary School Teachers." SIGCSE Bulletin, 5 (1973), 1:45-47.
22) "Computers and the Learning Society." Report of hearings before USHR, October 1977, US Govt Printing Office, 1978.
23) Conference Board of the Mathematical Sciences Committee on Computer Education. Recommendations Regarding Computers in High School Education. CBMS, Washington, D.C.
24) Conference on Basic Mathematical Skills and Learning. U. S. Department of Health, Education, and Welfare, Euclid, Ohio, 1975.
25) Dennis, J. Richard. "Undergraduate Programs to Increase Instructional Computing in Schools." Proceedings of Ninth Conference on Computing in the Undergraduate Curriculum, East Lansing, Mich, 1977.
26) Dennis, J. Richard, Dillhung, C. and Muiznieks, J. "Computer Activities in Secondary Schools in Illinois." Illinois Series on Educational Applications of Computers, No. 24, Univ. of Illinois, 1977.
27) Esterson, D.M. "Problems of Implementation: Courses for Pre- and In-Service Education". Information and Mathematics In Secondary Schools. North-Holland Publishing Co., 1978.
28) IFIP Technical Committee for Education, Working Group on Secondary School Education: Computer Education for Teachers in Secondary Schools: Aims and Objectives in Teacher Training AFIPS, Montvale, NJ, 1972.
29) Leuhrmann, Arthur. "Reading, Writing, Arithmetic, and Computing" in Improving Instructional Productivity in Higher Education. Educational Technology Publications, 1975.
30) Molnar, Stephen. "Computer Literacy: The Next Great Crisis in American Education.". Oregon Computing Teacher, vol 6, no 1, Sept 1978.
31) Moursund, David. "Report of the ACM Teacher Certification Subcommittee." SIGCSE Bulletin, vol 9, no 4, Dec 1977, pp-8-18.
32) Poirot, James L. "A Course Description for Teacher Education in Computer Science." SIGCSE Bulletin, vol. 8, February 1976, pp. 39-48.
33) Poirot, J.L. and Groves, D.N. Beginning Computer Science. Manchaca, Texas: Sterling Swift Publishing, 1978.
34) Poirot, J. L. and Groves, D.N. Computer Science for the Teacher. ,amcjaca. Texas" Sterling Swift Publishing, 1976.
35) Recommendations Regarding Computer in High School Education. Conference Board of the Mathematical Sciences, Washington, D. C., April 1972.
36) Taylor, Robert P. "Graduate Remedial Training in Computing for Educators." SIGCSE Training Symposium Proceedings, Dayton, February 1979.
37) Taylor, Robert P. (editor). The Computer in the School: Tutor, Tool, Tutee. New York: Teachers College Press, 1980.
38) Technology in Science Education: The Next Ten Years. National Science Foundation, Science Education Directorate, Washington, D.C., July 1979.
39) Topics in Instructional Computing. A special publication of SIGCUE, 1975.

Invited Session

CAUSE PROGRAM AND PROJECTS

Chaired By Lawrence Oliver
Program Manager CAUSE
National Science Foundation
Washington, D.C. 20550
(202) 282-7736

ABSTRACT

Information about the Comprehensive Assistance to Undergraduate Science Education (CAUSE) program and a brief historical overview will be presented with emphasis on projects involving the use of computers in education. Exemplary CAUSE projects will be presented focusing on: 1) individualized testing using microprocessors; 2) minicomputers in undergraduate laboratory science education; and 3) computer generation of instructional materials.

PARTICIPANTS (Listed in order of Presentation)

A Brief Historical Overview of CAUSE Computer-Related Projects

Larry Oliver
National Science Foundation

Individualized Testing Using Microprocessors

Douglas McCohm
University of California, Davis

Minicomputers in Undergraduate Laboratory Science Education

Elisha R. Huggins
Dartmouth College

Computer Generation of Instructional Materials

Michael P. Barnett
Brooklyn College of CUNY

Tutorial

DATA BASES - WHAT ARE THEY?

Arlan DeKock
University of Missouri-Rolla
325 Math-Computer Science Bldg.
Rolla, Missouri 65401
(314) 341-4491

ABSTRACT

This workshop will cover the three major approaches to current data base systems= hierarachical, CODASYL, and relational. The relative advantages and disadvantages of each approach will be compared. In particular IBM's Database System IMS, Cullinane System IDMS, and Cincom's System TOTAL will be discussed.

Computer Science Education

REQUIRED FRESHMAN COMPUTER EDUCATION IN A LIBERAL ARTS COLLEGE

David E. Wetmore
St. Andrews Presbyterian College
Laurinburg, N.C. 28352

(919)-276-3652x367

Computer education requires a definition of the group to be educated, the facilities (hardware and software) with which the education is to be accomplished, the goals of the educational process, and the level and rigor of that process. In practice, the first two factors often determine the last two.

On the undergraduate level, there are three fundamentally different groups to be educated: potential computer scientists, other science students, and everybody else. There is some agreement on the needs of the first two groups. The needs of the last group, the educated members of society who are not mathematically or scientifically oriented, are not so well defined. This paper describes the attempts, spanning a decade, of one institution to educate "everybody else."

St. Andrews Presbyterian College is a small (600 student) liberal arts college. In 1969 we instituted a science course required of all freshmen, "Selected Topics in Modern Science." This course is both the terminal course for non-science majors and the introductory course for all science majors. We felt that the ever increasing role of the computer in society required that we give all of our students a familiarity with computers. We decided that this could best be done by teaching them the rudiments of a programming language. At that time we had no interactive capabilities and all our work was done in batch mode. Software constraints limited our choice of languages to PL/I, PLC, or FORTRAN. Most of our students are not mathematically inclined, and so we chose PLC because of its string handling capabilities.

Over the next seven years, until the acquisition of interactive hardware, our computer education program evolved into a fairly stable system. As our present system is a direct response to our earlier experiences, a relatively detailed description of our old system follows.

The program consisted of eight three-hour sessions held weekly. The average section size was about 25, and there were usually six to eight class sections. During the first week there was a short introductory lecture, followed by the ideas of algorithms and flow diagramming. Students were given a simple billing problem, its flow diagram, and its program and were asked to punch up the program, run it, and bring their output to the next session.

At the second session the printouts were discussed, with emphasis on the debugging process. Then non-formatted I/O, assignment, and unconditional branching statements were introduced. The assignment for outside work was to modify the billing program to handle multiple customers by a loop and to write and run a program to calculate averages.

At the third session we introduced arrays and conditional branching statements. The students were to modify their average programs to use arrays and also to calculate the average deviation of their set of values. In the fourth session we held a review and then discussed comment lines and simple output formatting. The students' assignment was to rewrite their averaging program using comment lines and formatted output to increase the understandability of the program.

In the fifth week character strings were introduced. The assignment was to produce a program calculating the average word length in a text. The sixth week was a continuation of character strings. The homework assignment was to write and run a program converting clear text into pig Latin.

During the seventh and eighth weeks, each student was expected to design, write, and debug a program which he felt might be useful to him in his academic career. Most students wrote such things as checkbook balancing programs, grade point average calculators, or simple statistical packages.

This educational venture was successful in teaching students the rudiments of programming, the strengths and weaknesses of the computer and something of the potential role of the computer in society. We were not successful in convincing the students that the computer was a useful tool, however, as most of them never used it during the rest of their academic career.

In 1977 we purchased a Digital PDP-11/60 computer with eight video terminals for student use. During our first year with this system we did nothing more than modify our former system by switching to BASIC and using the "TUTR" series to teach the language. The homework assignments remained unchanged. Our work convinced us that there are certain attributes of an ideal curriculum for teaching everybody else about computers. I believe that there are six such attributes:

First, what is being done must be of obvious applicability to the student. Many college freshmen are not mathematically inclined and do not foresee any use to them of the computer as number cruncher.

Second, the system must obviously be computer dependent. In our old system, none of the assigned programs would repay the effort involved in writing and debugging the programs.

Third, the material must be capable of gradual entry. New knowledge should be required in small increments. Although our original sequence was modified quite frequently, we were never able to avoid one session in which the students were overloaded with information.

Fourth, the system should be word-oriented instead of number-oriented. Most students are more comfortable with words than numbers. For example, we found that interest in the computer increased dramatically when we started to work with character strings.

Fifth, the system must have understandable error messages. For beginning students a cryptic error message is probably worse than a simple statement that an error has been committed.

Sixth, the system must possess the attributes of any good tool: versatility, usefulness, resistance to breakdown, obvious efficiency, and feedback to the user.

We decided that the system which would best combine the attributes listed above was word processing. Word processing includes programs to write text, alter text, move text within or between files, and to format text. The systems are not only obviously computer dependent, they are impossible without a computer. It is apparent that they will work with textual material of any length or complexity.

The major advantage of word processing is its applicability. A freshman at a liberal arts college knows that he will write many papers over the course of the next four years. Any tool that will make that writing easier is welcomed.

Word processing systems are usually designed for use by non-technical people and are structured so that much can be done with a few commands. Additional material can be learned when its need is obvious to the student. There are no quantum leaps in knowledge requirements. Furthermore, because of the users for which most word processing systems are designed, they are quite fail-safe, and their error messages are usually clear and self-explanatory.

We chose EDT, a line-oriented editor as our editing system. EDT is not as powerful as other available editors, but it is characterized by simplicity and ease of use. RNO was chosen as our formatting program. It is capable of straightforward formatting with a very few commands but can be used for extremely complex formatting jobs.

The textual material used in the course is a 25-page manual consisting of the following information:

- Introduction
  - Logging on
  - Account security
  - Overview of the programs
  - Logging off
  - Files and file naming
- Editing Program
  - Entering the program
  - Creating files
  - Writing files
  - Appending to files
  - Error messages
  - Editing, an overview
  - Text display
  - Finding text
  - Adding lines within text
  - Deleting lines from text
  - Changes within lines
- Getting Hard Copy
- Text Formatting
  - Introduction
  - Case control
  - Margin control
  - Centering text
- Displaying Files
- Account Directories
- Spelling Checker Program
- Summary Table of Programs

Each new student at St. Andrews is assigned a computer account. To use this account it is necessary only to go to the computer center and obtain the account number and password. Each May, accounts are deleted unless the student requests that it be kept. At present 230 students (37% of the student body) have active accounts. About 22% of the students not taking a computer science course or the freshman course have active accounts.

The computer block lasts four weeks with four assignments. During the first session the introduction of the manual, editing program through appending to files, and getting hard copy are covered. The students' first assignment is to activate their accounts and write a file of at least 100 lines in two different sessions. The student must hand in a hard copy of the file the next week.

For the second week the reading material is the remainder of the editing program section and the text formatting material. This material is discussed and questions are answered. The homework assignment for the third week is to edit the file produced during the first week so that it will include formatting commands for case and margin control and for paragraphing. A copy of the edited file is to be handed in.

During the third meeting the remainder of the manual is covered, and each student is asked to bring to the fourth session an output file (formatted text) from the file produced during the second week.

The fourth and final session is devoted to questions from the class and work with individuals. The homework assignment (which is weighted twice as heavily as any of the others) is to hand in, within three weeks, a copy of a computer-produced paper submitted for grading in another class.

An advanced word processing document is available in the college bookstore. The advanced document deals with additional features of the editing and formatting programs. This last year 174 students bought the required introductory text, and about 50 copies of the advanced text were sold. (*)

Student evaluation of the computer block has been favorable. In our usual term-end evaluation it received an average rating of very good. In past years, prior to word processing, the computer block was usually rated as good to fair. Furthermore, conversations with faculty members outside the sciences indicate that an increasing number of papers in upper level courses are being computer produced. Thus, although we have no hard data, we feel that we are having some success at our primary goal, that of showing all students the utility of the computer as a tool.

In addition, we feel that word processing is an excellent computer application with which to show students the usefulness, strengths, and weaknesses of computer systems.

*****

(8) Copies of the introductory handout may be obtained by sending a self-addressed, self-stamped ($0.70) large manilla envelope to the author.

DEVELOPMENT OF COMMUNICATIONS SKILLS IN SOFTWARE ENGINEERING

John A. Beidler
John G. Meinke

Department of Mathematics/Computer Science
University of Scranton
Scranton, PA 18510

(717) 961-7428

INTRODUCTION

How much will an individual's programming skills be enhanced by an ability to present ideas orally and in writing? There is much more to programming than just writing programs. Yet the emphasis in computing and information science curricula is on programming with little emphasis on many of the other essential tasks that surround and are included in the programming process. Since 1975, we have been teaching a course that provides a broader view of software engineering. This course, entitled Computer Projects, is an undergraduate degree requirement that is taken in the senior year. As part of this course, students develop and document an approved project of their choice and give several oral presentations as well as several written reports.

Normally, these projects begin during the junior year. The projects can be done in a variety of areas but must have a faculty sponsor. Students are encouraged to seek out faculty from other departments who are interested in using computing resources in some way to expand on existing projects. In any case, all projects must have faculty sponsors who are interested in the projects.

A key element in this course is its organization. During the semester, each student must give two lectures about his project and write seven reports. The combination of the oral presentations with the written reports produces a total description of the project and makes the students aware of a more global picture of the programming process as well as giving him a better understanding of the types of things that programmers and systems analysts do besides grinding out code.

Subsequent sections of this paper describe the course and the rationale for various items that are included and the reasons for the way the reports are organized.

COURSE ORGANIZATION

The course is organized around the presentation of student projects. For the first two weeks the students are given a general orientation regarding what is expected from them. During the next three weeks the students make their first oral presentations which must be between 15 and 25 minutes long and provide appropriate general descriptions of the project areas. The presentation should be directed toward convincing the audience of the importance of the project and the impact it has on its environment.

The middle three weeks of the course are spent describing the various methods that can be used in the remaining reports and the second oral presentation. Also, the students are shown the interconnections that should exist between the oral presentations and their various reports. The second half of the semester is devoted to the second oral presentations. These are 40 to 50 minute presentations on the technical details of each project. This talk emphasizes the appropriateness of the technique selected to solve the problem.

While the above sequence is occurring in class, the students submit reports about every two weeks. As a collection, these reports are supposed to form a complete description of the project.

THE ORAL PRESENTATIONS

Both oral presentations are evaluated by both students and faculty members present. Students are graded on a variety of factors such as content, the use of visual aides, and the appropriateness of the level of presentation for the audience. The first presentation is a 20 minute survey of the area of the project.

The purpose here is to acquaint the audience with the problem area and convince them of the importance of the project.

The second presentation is a 40 to 50 minute presentation on technical aspects, hardware, data organization, and algorithms. Part of this presentation must include, when appropriate, a structured walkthrough of a part of the project.

REPORTS

The seven reports, when taken as a unit, should form a complete description of the project. Content of these reports is a compromise between an ideal and what is realistic within the time frame of a semester. In all cases, the reports are judged not just on their technical merits but also on the organization, grammar, and spelling.

Report One. This report is an extended abstract of the project and should be about three to five typed double-spaced pages long. The students are told to relate this report to their first presentation. Specifically, the first talk should elaborate on the general problem area while the extended abstract should only briefly mention it.

Report Two. This three page report is a requirements analysis. Here the student must describe the hardware needs of his project and describe how he came to his conclusions. To help the students, a list of general questions is distributed. This list includes:

1. What resources does the project require? (Primary memory, disk, tapes, etc.)
2. Time requirements - What effect do various variables have on the time requirements?
3. Terminals - CRT, hardcopy, graphics, high quality hardcopy
   -Which is ideal?
   -Which are satisfactory?
   -Baud rates?
   -Synchronous, asynchronous?
   -Intelligent terminals, etc.
4. Program space requirements?
5. Backup

Report Three. This is a justification document. The students assume that they are writing a document in response to a request by corporate management for justification of continued funding of the project. Here the students are required to defend the value of their project over the status quo. Explanations of details must be given in a way which would be understood by a semi-technical manager or executive. As a suggestion, students are told that they should produce figures that describe their project versus the status quo, then emphasize how these data are arrived at legitimately.

One purpose of these first three reports is to give students a chance to collect their thoughts during the first five weeks of the semester. The rest of the reports demand a considerable effort on their part and require a thorough analysis of their projects.

Report Four. This report is a collection of four analysis and design documents. The first is a systems chart in the form of a tree structure that describes the top-down organization of their system. Included with this chart is a brief description of the purpose of each module in the system.

The second part of this report is a data chart. The data chart is in three parts--input, process, and output. The input section describes the files used as input to the system along with the record formats of input files. Each item in a record is defined in terms of its type and what role it plays in the system. The output section describes both the output file and update files. By an update file we mean a file used both as an input and output. As in the input section, all record formats and items in the record must be completely described.

The process section describes the type, bound, and purpose of every major variable used. This section is grouped first by describing global variables, then common variables, then a break down of other variables by module. By major variables, we mean that, for the sake of brevity, the students can selectively decide to limit the size of the data chart by not including all variables, for example, temporary storage and loop indices.

The third component in report four is a data flow graph. Again, due to time limitations, the students are not required to create a data flow graph of the entire system. Rather, they use several important items and show how they are transformed and combined with other data to produce the results. The data flow graph completes the systems chart in that the systems chart shows the tree structure of a system.

The fourth item in report four is a bibliography. This bibliography does not just state the references. Instead, a brief statement must accompany each reference describing the relevant information obtained from that reference. Also, the bibliography does not list only texts and periodicals. It also includes references to individuals who have provided information or assistance. It is grouped as follows:

1. Text References
2. Popular Periodical References (*Time* to *Creative Computing*)
3. Technical Periodical References (all other periodicals)
4. Individual Citations (persons who assisted)

With report four, we have a broad based view of the project. This report acts as a framework upon which the rest of the reports can be built and integrated.

Report Five. Report Five is a user's manual, *usually* the most difficult report for the students to write. The fundamental problem is that they know too much about their project, and they lack an appreciation of the difficulties that others encounter (especially those who are not experts in computing) in simply using the project.

The key to improve this report is an emphasis on well-constructed examples. Even this is a problem, however, because the students usually create unusual esoteric examples that are not typical of normal use of their project. That is, their examples are often like the horrible examples we find in manuals that are produced by the various computer manufactures.

Another problem that is evident in this report is poor grammar. In order to correct this with a minimum effort, the students are told that the report is graded using the basics in Strunk and White's *Elements of Style*. Also, some time is spent on some of the basics.

A final emphasis in this report is the importance of stating everything. That is, many times students assume that the readers of their reports have the same background and knowledge that the authors have. As such, their report is not complete because there is still a substantial amount of knowledge that is locked up in their heads and not stated in black and white.

As a complement to report four, this report explains the details, critical elements, and various interrelationships in the system. Here, students are encouraged to find the weaknesses inherent in the various graphs and charts of report four.

Report Six. This is usually the easiest report for the students to write. They are so wrapped up in the technical details of their project that the volume of material that can be included contributes significantly to the generation of this report. However, there are still problems they encounter in construction of this report.

Their biggest problem is one of organization. The students are often so wrapped up in the project that they don't know where to begin writing. Surprisingly, they find it difficult to use the sections of report four as an outline for this report. Once they realize the usability of report four in generating report six, this report becomes well organized and easy to read.

Report Seven. This report contains four sections. The first section consists of annotated and documented program listings. The documentation should explicitly indicate the connections between the code generated for the project and the various reports. Section two is a set of corrections to reports four, five, and six, which allows students to modify these reports to stay in line with the way their system has evolved.

Section three of this report is another difficult section. Here students are required to make a critical evaluation of their system, pointing out flaws and deficiencies both in their overall design and implementation. This is followed by section four, which describes what was learned through the project. Included in this section are recommendations for modifications, enhancements, and ideas for future projects.

OBSERVATIONS

There is no doubt that this course gives the students a broader appreciation of the entire hardware/software development process. If there is a flaw, it is not in this course; rather, it is that preceding courses do not require more of the development process so that this course will be less of a shock to students.

Typical of the variety of projects that students have done in this course are:

1. Development of relative and hierarchical data base simulations
2. General purpose software science measurements system
3. Programmable calculator use in the physics laboratory

4. On-line accounting laboratory
5. An enhanced editor
6. Graphics: Tektronics - Diablo interface
7. Graphics: 3-D perspectives with hidden line elimination
8. Software for forcasting the university's financial status

By allowing a broad collection of projects, we have obtained an interesting by-product. Faculty from other departments, as well as various administrative offices, are more aware of the potential of our computing resources. Therefore, despite a stable, or slightly shrinking student population, we have created an increased demand for computing resources. This has given us an opportunity to obtain the variety of computing resources that we need to support our computer science program.

SYSTEMATIC ASSESSMENT OF PROGRAMMING ASSIGNMENTS

Judy M. Bishop
Computer Science Division
University of the Witwatersrand
Johannesburg 2001
SOUTH AFRICA
(tel. (11)-394011)

SUMMARY

The obvious way to assess a course on programming is to assign marks to programs written by the students. However, it is not evident at the outset that systematic assessment is feasible. Nor is it necessarily apparent what features the marker should look for and the weight to be attached to them. This paper explains a method that has been used for four years at two universities to mark course work at second year level. The experience gained shows that systematic assessment is possible and can be used to provide guidance to students as to what constitutes a good program.

BACKGROUND

The second year course in advanced programming [Barron 1975] is the backbone of the computer science degree at both the University of Southampton, England, and the University of the Witwatersrand, South Africa. Since 1974, it has been taught with the aim of turning out not just good, but excellent programmers. Like Noonan [1979], we felt that the primary student product of such a course should be a single, large programming project, split into four smaller projects. Two such projects were designed to be used in alternate years, each requiring a total of about 1000 lines of Pascal [Mullins 1974, 1975]. Although a written examination is required to test a full knowledge of data structuring and sorting techniques, the bulk of the final mark for the course is derived from the project. At the outset, therefore, it was realised that the marking of these projects had to be of a higher standard than usual.

Very little guidance in grading can be found in the literature. Abshire [1978] in his otherwise excellent advice for teachers, skirts the problem, simply stating "Establish a grading policy ... Grade each program ... Tell the students your grading policy." In particular, how does one neutralize the effect of subjective factors such as the student's previous performance, his presentation (neat or messy), and the preference of different markers for one or another style of program layout?

The methods used to mark the assignments for large first year classes or small third year classes do not seem approriate for a medium sized second year class (40-80 students). Program assignments in a first course can, for the most part, be assessed by checking that they work for certain samples of data [Noonan 1979] although some teachers avoid this approach and assess only by a special form of examination [Radue 1976, Trombetta 1979]. At the other end of the scale, projects undertaken by third year and graduate students can be looked at individually and marked over a wide range of criteria [Hall 1979]. Second year assignments fall somewhere in the middle. One needs to give marks for good program and data design as well as for correct results, but the numbers involved preclude spending hours agonizing over each program. In fact, the projects as set up require the first three parts to be handed in at two week intervals. Each part builds on the next so that it is desirable that work handed in on Friday be marked and returned by Monday. So as not to rely on the impossible, and also because typically some 40% of solutions are not handed in in final form, specimen solutions are distributed and students are free to use these instead of their own in the next phase. Nevertheless, speed is a factor.

As a final complication, the marking may be shared between two or three people and this means that a standard must be well defined and understood. These three requirements of quick, accurate, and equal marking led to the design of the systematic method presented here.

THE METHOD

The method quite openly borrows from a technique used in Pascal to make programs more readable. Instead of working in numbers, a marking scheme is set up using words or phrases to describe the assessment of the solution under various criteria. Once all the solutions have been marked, weights are assigned to the words and a final mark calculated. This mark is then compared to the earlier estimates made on impressions and the marks might be moderated one way or the other. The six steps are:

1. Divide the solutions into five groups based on evident output.
2. Set up a symbolic marking scheme.
3. Mark all the solutions according to the scheme.
4. Assign weights to the words used in marking.
5. Calculate a numeric mark.
6. Compare the estimated and calculated marks and moderate if necessary.

1. Rough Grouping

Divide the programs into groups according to the evident output.

A. Fully correct
B. Answers partially wrong
C. Incomplete output, perhaps with an execution error
D. No output - execution error in reading phase
E. Compilation errors

These groups can be roughly translated into the traditional classes as follows:

A	: First	(80%+)	(unless layout is horrible or algorithms messy or inefficient)
B,C	: Second or third	(50%-79%)	(the largest group - should not get a First or Fail)
D	: Third or fail	(59%-)	(tricky group - they must be marked on criteria other than results)
E	: Fail	(49%-)	(depending on the level of the course, these might get zero)

2. The Marking Scheme (Grading Policy)

A marking scheme is devised by selecting five or more criteria on which the programs might differ, for example, the algorithm, design of output, achievement, and testing. In initial assignments, the criteria would emphasize points such as neat program layout or the correct use of procedures and parameters. In later assignments, a quick glance at the solutions handed in will show whether these lessons have been learned and are no longer worth assessing because the standard is uniformly high.

Each criterion is then assigned a list of between two and five possible values. These are meaningful adjectives, applicable both to the criterion and to the expected results. A typical scheme would be:

answers = (right, some wrong, none, still reading)
output design = (as required, improvement possible, shortcuts, unformatted)
algorithm = (elegant, competent, average, messy)
achievement = (more than enough, fullfilled requirements, gaps left, far too little)
testing = (complete, partial, inadequate)
self-contained = (yes, no)

Some comments on this scheme are:

1. If there are no answers, the potential output design will still be assessed.
2. In a text formatting problem, "shortcuts" would be given if, for example, formatting such as centering or right-justifying was done by spacing to an absolute column number instead of being based on the length of the line. "Unformatted" would be granted when there is no centering or justifying. Possible improvements would be small things such as leaving a line at the bottom of a page before printing the page number. The rating of the output design should not be confused with that for achievements.
3. The values given here for algorithm have been found to be applicable only when the marker is genuinely astonished at the standard. A competent algorithm is one in which
   - there are no redundant or out of place variables
   - control structures are used correctly (e.g. case, not cascading if's)
   - loops cover short ranges (i.e. if a loop is more than 20 lines long, part of it should be in a procedure)
   - the intention of each line is obvious, with comments only being necessary to indicate yet-to-be-included sections or modifications that might be considered.
   - the use of the language is quietly apt, avoiding over-simplification and clever tricks alike.

   Most students' algorithms fall down somewhere and are classed as average. A messy program can be recognized a mile off!
4. At some stage, one wishes to emphasize that procedures should be written, as far as possible, as self-contained entities. This does not just mean that they should have local variables, but that they could be called from different contexts and still produce the same results. This discipline is particularly important in a course where one assignment leads to the next. If procedures are not self-contained, they will have to be altered before being lifted out of

one program into another.

5. The testing criterion is there to get students out of the habit of handing in only one run. Very often instructors prefer a single run because it is less to mark, but in fact the opposite is true. It is easier to put the onus on the student to illustrate what his program can do rather than have to deduce this from the listing.

Careful consideration must be given to the interaction of individual criteria. A prime example is that of testing versus achievement. If a feature is included, such as a test for an unexpected end-of-file, then a run showing that it works should be handed in. This can lead to a program being given

achievement = fulfilled requirements
testing = partial

if the end-of-file was checked for, but only a run with correct data was handed in. On the other hand, a program that did not make the check would get

achievement = too little
testing = complete

because it had actually tested all that it could do. The weightings for these criteria (Step 4) would be assigned such that the first program would get slightly more marks.

3. Marking (grading)

The process of marking uses two tables. On the first, values for the criteria in the marking scheme are filled in for each student, with the last two columns, "mark" and "calculated class", being left blank for the time being. On the second table, the group is recorded, alongside an "estimated class" and a comment. The estimated class represents one's gut reaction to the students' efforts, and is as far as many informal marking systems go. The comment lists any relevant factors that would not show up in the marks and could be used at a later date to explain border-line cases. Typical table entries would be:

TABLE I

NAME	ANSWERS	OUTPUT	ALGORITHM	ACHIEVE	TESTING	SELF-C.	MARK	CALC.CLASS
J. Mullins	right	as required	average	ful.req	partial	yes		

TABLE II

NAME	GROUP	EST.	COMMENT
J. Mullins	A	I	Good use of the language

Two further lists are useful. One records the names of students whose programs are worth distributing and notes the points that the programs illustrate. The other lists the important points that should be commented on in tutorials or in a handout, including common misconceptions or interesting approaches.

4. Weights

Only at this stage are numerical values assigned to the words used in classifying the students' solutions. Each value in each criteria is assigned a weight, so that the left hand values add up to the desired maximum mark. For most assignments, a maximum of 20 is adequate. Suitable weights for the sample scheme above are:

```
             5        3       1         0
answers = (right, somewrong, none, still reading)
                     5
output design = (as required,
                         3
                 improvement possible,
                      2          1
                 shortcuts, unformatted)
               6          5         3        1 (2)
algorithm =(elegant, competent,  average,  messy)
                      6
achievement = (more than enough,
                      5                    3
               fulfilled requirements, gaps left,
                      1
               far too little)
              4         3          1
testing = (complete, partial, inadequate)
                  1    0
selfcontained = (yes, no)
```

The maximum mark in this scheme is actually more than 20 because elegant algorithms and achievement more than required should be rewarded by additional marks. These scores could well be cancelled out by the student falling down on some other aspect, such as testing. Note that even the worst program must get some marks (for effort) so that zero does not appear for every criterion.

5. Classifying

Simple summations give the actual marks for the first table. From these, classes are assigned as follows:

CLASS	%	MARK
I	80+	16+
II-1	70-79	14-15
II-2	60-69	12-13
III	50-59	10-11
F	20-49	5-10
FF	0-19	0-4

6. Moderating

In this final step the effort so far is

rewarded by comparing the three assessments:

- the rough grouping based on visible results
- the gut reaction class estimate
- the mark calulated on detailed criteria.

If there are violent disagreements between the estimated and calculated classes, the weightings should be adjusted first. Adjustment was made in the above scheme because a program with a very messy algorithm got 16 marks and really should not have got a first. Therefore messy was given a weighting of 1 instead of 2.

	I	II-1	II-2	III	F
Rough	12	(	24 combined	)	9
Estimated	11	5	13	5	7
	↙ 1		↘ 1	↘ 4	↘ 3
Calculated	12	3	10	6.	10

The above table shows a comparison from an actual assignment given to 41 students. Only one program improved its rating on the calculated mark, while eight were demoted. The value of the calculation method is evident in that close examination showed the estimates to be faulty. Three programs in Group C had tiny errors and would have worked soon. They were estimated at II-2s. Then the scheme showed that they had shortcut or unformatted output in addition and correctly placed them in class III. On the other hand, another C program had top marks for output, algorithm, and achievement which compensated for the slightly wrong answers and moved the class from a II-1 (estimated) to a I (calculated).

EVALUATION

Although the method takes a while to describe, in practice it is a stream- lined process which can enable 60 assignments of about 300 lines each to be accurately assessed in about twelve hours. The setting up of the scheme takes about twenty minutes and assigning the weights about ten minutes. The grouping, classifying, and moderating take about an hour. Given this investment, the individual solutions can be marked in ten minutes each. This compares favourably with the norm of thirty minutes for an examination script.

An important advantage of the method is that it almost eliminates prejudice of any sort. The example above was taken from the experience of a marker who is usually quite generous and showed that some 20% of the estimates were too high. On the other hand, there are some markers who equate "very good" with 7 out of 10 and have a mental block against giving full marks. With this scheme, they find that they can justify "complete" for testing in their minds and let the calculations take care of themselves.

One of the criticisms of the method may be that it does not encourage a fine enough net. In nearly all cases, the weights in the above example jump from 5 to 3 to 1. This is intentional. After four years experience, we have found that one cannot make the fine distinction between an algorithm that is excellent, very good, good, fair or poor. These terms are too vague and are avoided in favour of competent, average and messy, with which one can identify a program more readily. Thus the marks do tend to be discrete and categorized rather than continuous.

Although the method does not provide any built-in checks against plagiarism, the fast pace with which programs can be marked means that one's memory is capable of detecting striking similarities. In four years, one case of genuine plagiarism was detected and proven. Numerous similar programs were found, but by referring to the scheme, it was established that they differed in one or more important details.

Using the method has definitely increased our understanding of what constitutes a good program. As a result, our teaching has improved and certainly the standard of the student programs has risen over the years to such an extent that we would disagree with Deimel and Pozefsky [1979] who state that "Student-written programs accepted by computer science instructors are usually inferior to programs which exemplify currently-accepted 'good' professional practice." Our experience is that students produce programs which are just as good, if not better, than those in an "up-to-date programming shop".

Because the symbolic scheme does not have actual marks, it can be shown freely to students so that they know what to aim for. The tables of detailed evaluations are also invaluable in countering queries about results because the words are all there to explain why the student lost marks (e.g. shortcuts, inadequate testing).

CONCLUSION

This method of grading is ideal for medium sized classes where programming excellence is examined by practical work. It minimizes the effects of prejudice and provides a balanced assessment of the factors that are being particularly stressed at any one phase of the project. It is adaptable to most types of projects and has been expanded and used with success in more senior classes. It increases ones understanding of good programming and enables one to communicate this to students in a handy way. Finally, and most importantly, it establishes confidence in systematic assessment for both staff and students.

ACKNOWLEDGEMENTS

The Advanced Programming course was devised by David Barron of the University of Southampton and it was at his suggestion that this method of grading was designed. Thanks are due to the

classes of 1975 to 1979 who responded with such enthusiasm to these ideas.

REFERENCES

Abshire, Gary, "Techniques for Computer Science Teachers", SIGCSE Bulletin 10 (4), 42-46, December 1978.

Barron, D.W., "Design and Construction of Computer Programs: A Course in Advanced Programming", Computer Studies Group, University of Southampton, 1975. Revised 1978.

Deimel, Lionel and Pozefsky, Mark, "Requirements for Student Programs in Undergraduate Curriculum: How Much is Enough?", SIGCSE Bulletin 11 (1), 14-17, February 1979.

Hall, Colleen, "Third Year Project Marking Scheme", Computer Science Report, University of the Witwatersrand, July 1979.

Mullins, Judy, "The Family Tree Project for Advanced Programming", Computer Studies Group University of Southampton, 1975. Revised at Computer Science Division, University of the Witwatersrand, 1978.

Mullins, Judy, "The Play Structure Project", Computer Studies Group, University of Southampton, 1976. Revised at Computer Science Division, University of the Witwatersrand, 1979.

Noonan, Robert, "The Second Course in Computer Programming: Some Principles and Consequences", SIGCSE Bulletin 11 (1), 187-191, February 1979.

Radue, J.E. "On the Teaching and Evaluation of a FORTRAN Service Course" SIGCSE Bulletin 8 (2), 32-35, June 1976.

Trombetta, Michael, "On Testing Programming Ability", SIGCSE Bulletin 11 (4), 57-60, December 1979.

DATA STRUCTURES AT THE ASSOCIATE DEGREE LEVEL

Richard F. Dempsey
Computer Science
The Pennsylvania State University
The Worthington Scranton Campus
Dunmore, Pa. 18512
717-961-4757

Recent design methodologies based on the structure of the data and the development of data base management systems which use a wide variety of data structures add weight to the importance of data storage and processing. The fundamental role played by data structures courses in applied bachelors degree programs also indicates its significance (1, 2, 3). Associate degree programs that train students to be quality entry-level programmers with sufficient background to adjust to new trends in data processing must provide these students with background in both internal and external data structures (4). This paper describes the philosophy and content of such a course taught as part of the associate degree computer science curriculum at the Scranton Campus of Pennsylvania State University.

The objectives of the course are as follows:

1) To present the structure and functional characteristics of external storage devices and their impact on file organization techniques.
2) To present concepts of internal and external sorting and searching techniques.
3) To present the advanced COBOL language elements associated with the above.
4) To present the concepts of internal data structures.
5) To expose the student to the use of libraries and utility routines.
6) To make the student aware of considerations in the design of data files.

This course, titled "Techniques of Organization," is offered in the first term of the students' second year. It is a three-hour course. Prerequisite courses from the first year are:

1) 1st term. "Introduction to Algorithmic Processes." This course emphasizes solving problems using the computer as a tool. The language is either WATFIV or PL/C.

2) 2nd term. "Computer Organization and Programming." This course emphasizes the binary and decimal instruction set of the IBM 360/370 Assembler Language. The objective is to get the student to see how the computer operates at machine level. This provides a solid background to make the student a better COBOL programmer and debugger. All programs are run on ASSIST [described in (5)].

3) 3rd term. "Introduction to Data Processing." This is not the usual course using this title, but a high-powered beginning COBOL course. As the students have programmed in two languages already, this course moves fast. It covers such topics as table handling, SORT verb, sequential files on disk and tape, and REPORT-WRITER. Emphasis is placed on programming style, documentation, and quality of code. Structured programming is emphasized throughout. The programs are typical data processing problems.

The students now have a solid, basic programming background. Through the "Techniques in Organization" course, this background will be reinforced while they learn to handle data in new ways. The material in this course is presented with an applications orientation and a minimum of theory. The objective is to give the student a working knowledge of data organizations and experience at using them. Some topics will simply be introduced and reinforced in following courses.

The order of presentation of the topics in this course is not always the same. The nature of the programming assignments, shifts in emphasis due to new trends in data processing, and the makeup of the class itself all affect the arrangement of topics. These topics, with approximate times in parenthesis, are:

Topic 1. Indexed Sequential Files (1.5 weeks) A thorough working knowledge of ISAM is expected. Some time is spent on the physical aspects of ISAM files so that the students are aware of what is physically happening. The pros and cons of ISAM files and the options available within it are discussed. The elements of COBOL needed to process ISAM files are presented.

Topic 2. Direct access files (1 week) Both relative and direct files are covered. The concept of hashing is introduced. The pros and cons of direct access files are discussed in relationship to the other file organizations. The COBOL elements needed to process these files are also presented.

Topic 3. VSAM files (1 week) VSAM files are discussed, with major emphasis on the KSDS format. The physical characteristics are presented and compared to those of ISAM. Comparison is also made at the COBOL language level between ISAM and VSAM.

Topic 4. Sorting (1 week) External sorts are covered briefly. One or two techniques are looked at so that the student can see the nature of what takes place in an external sort. No attempts are made to teach the student how to write an external sort. Internal sorts are also discussed. In the first year the student has programmed at least one internal sort, such as the bubble sort. Discussion here is held to the relative efficiency of such sorts in relationship to data set size. A sorting technique, such as Quicksort, is presented as an example of a sort for larger, internal data sets.

Topic 5. Internal Data Structures (3 weeks) Problems are discussed that emphasize some of the limitations and problems resulting from using only the basic internal storage techniques covered in the language courses. This discussion leads into a working level presentation on linear lists, stacks, queues, singly and doubly linked lists, trees, and inverted lists. Algorithms for handling some of these factors, as well as ways of manipulating and allocating storage for them in COBOL are examined.

Internal and external data structures are compared. For example, a good tie-in of the similarities and differences would be an investigation of how a set of records that need a tree structure relationship can be stored on external files using various file organizations. By this time the students are getting a good grasp of data organization and its impact on programming efforts, which should provide them with a good basis for what is happening in most of the data base management systems they will surely face. Such knowledge is essential to making effective use of these systems.

Topic 6. Searching (1 week) A short time is spent on various searching techniques as they relate to the various data structures. This discussion further enhances the students' understanding of the significance of storing data in different forms.

Slotted with these six main topics are some others, basically for programming purposes. Partitioned data sets are introduced to allow the use of libraries and the COPY verb. These libraries are created for the students, but the JCL and control statements for IEBUPDTE are explained. The JCL and COBOL statements to handle subprograms are also presented.

The programming assignments for the course are all done in COBOL. Students are given all needed JCL, but all elements of it are explained so they see the reason for each element of each JCL statement. All programs must be structured and completely documented. They are evaluated on quality and correctness.

Typical programming assignments are:

Prog. 1. An input edit routine that creates a sequential file on disk of the valid records. This file is then sorted in a second step by calling the system SORT/MERGE package via JCL.

Prog. 2. An ISAM file created from the output of Prog. 1. Libraries and the COPY verb are introduced.

Prog. 3. A report produced by sequentially processing the ISAM file from Prog. 2. This assignment introduces the use of the IEBISAM utility to allow keeping an ISAM file as a sequential file. (Students are not allowed to keep permanent files on our system).

Prog. 4. A random update, including adds, deletes, and updates, processed against the ISAM file from Prog. 2.

Prog. 5. A three-step program that creates, updates, and produces a report from a relative file.

Prog. 6. A program to implement the Quicksort algorithm discussed in class. This assignment requires the use of COBOL subroutines and stacks.

Prog. 7. A program to build and manipulate a singly linked list.

Some of the topics, such as direct files and trees, are not implemented in this course. The next course, covering advanced assembler, advanced debugging, utilities, and JCL, provides an excellent place for students to write programs using these concepts.

This course covers a lot of material in a ten-week term. Care must be taken not to get too deep into topics that don't merit the attention in relationship to other topics. Creating more than one course for the material is difficult as there are only six basic computer science courses plus a projects course in our curriculum. This limited number of courses results from the University's strong belief in the total education of the student and concurs with the guidelines set forth by the ACM SIGCSE paper on associate degree programs (6).

A major problem with this course is the lack of a suitable textbook to cover such a broad range of topics. Most books only cover a subset of these topics and even then tend to be too elementary or too theoretical. The topic of internal data structures has proven to be a significant problem in this area. Barrodale et al (7) has a nice approach for our applications level but is not detailed enough; most other data structures books are too detailed and theoretical.

This course has kept many computer science majors off the streets a night or two, but the material and programs have provided a solid background for our graduates. They become productive almost immediately upon employment in such areas as application programming and technical support. Several have been able to use the course as a springboard into data base systems. This practical applications-level study of the area of data storage and organization has given them the experience and confidence to be quality members of the data processing field and to adjust quickly to the changing demands of their field.

1. Fosdick, Howard and Karen Mackey, "A Course in the Pragmatic Tools of the Programming Environment: Description and Rationale," SIGCSE Bulletin, Vol. 11, No. 3, Sept. 1979, pp. 11-13.
2. Mackey, Karen and Howard Fosdick, "An Applied Computer Science/Systems Programming Approach to Teaching Data Structures," SIGCES Bulletin, Vol. 11, No. 1, Feb. 1979, pp. 76-78.
3. Beidler, John and John Meinke, "A Software Tool For Teaching Data Structures," SIGCSE Bulletin, Vol. 10, No. 3, Aug. 1978, pp. 120-122.
4. Little, Joyce Currie, "Computer Education and Community Colleges," Interface, Vol. 1, Issue 1, Winter 1979, pp. 12-16.
5. Overbeek, R. A. and W. E. Singletany, Assembler Language with ASSIST, Chicago: Science Research Associates, 1976.
6. Little, Joyce Currie et al, "Curriculum Recommendation... in Computer Programming," SIGCSE Bulletin, Vol. 9, No. 2, June 1977, pp. 17-36.
7. Barrodale, et al. Elementary Computer Applications, Wiley, 1971.

Integrating Computing into K-12 Curriculum

THE SCARSDALE PROJECT
INTEGRATING COMPUTING INTO THE K-12 CURRICULUM

Thomas Sobol, Superintendent of Schools, Scarsdale, N. Y.
Robert Taylor, Teachers College, Columbia University, N. Y., N. Y.

Introduction

Within a few years every child in America is likely to have at least one personal computer. The potential impact upon schools staggers the imagination. At the least, it is likely to move the focus of education from end product to process and raise visual and auditory forms of information to a status rivalling that of written language. Because ideas can be presented, explored, and expanded by human interaction with the computer, computing is certain to transform the schools from kindergarten upwards; its impact will be as broad and deep as any intellectual innovation in recorded history, including printing. In addition to traditional communication, teachers and pupils will communicate through the computer immediately and dynamically, in word, picture, and sound, with each other and with others throughout the community whom they will never meet face-to-face. Thus the nature of the communication will be transformed and the range of participants will be startlingly enlarged.

This stage of mass computer use was forseen years ago; it arrives at last because computers have been improved, made smaller, and produced considerably cheaper over the last five years. Yet many of the questions raised in the minds of those who foresaw what was coming remain unanswered. How will use of computers affect the child's development? What are appropriate languages for young

166

children to use in talking to computers? What devices other than the traditional keyboard and screen or terminal/printer can, when attached to a computer, maximize its educational utility for the young child? What should teachers know about computers? How expert should a teacher be in computing to make reasonably effective use of it in teaching and learning? What is the teacher's role in a classroom where every student has free access to seductively engrossing computer power? What is the impact of computer games on the child and what role should such games have in the school? When should the computer be used as a tool, when as a tutor, and when as a tutee?

Little significant research has been done on most of these questions, even by pioneers in computing and education. The research that has been done focuses on a narrow subset of the issues such as the effectiveness of computerized drill and practice in arithmetic or spelling. There are few clear answers so far; often what are advanced as answers are little more than strongly held hypotheses. There is no ready-made curriculum, even in outline; there are no coherent texts; there is little available teacher training; and there is little public understanding of what changes computing might promote in education.

Given all these unknowns, if a public school system takes the transforming impact of computing seriously, where does it begin? Who does what with whom, at what cost and with what type of success? Though little research has been performed, though many more questions have been raised than have been answered, and though nothing like a comprehensive intergration of computing into the curriculum has been formulated, the computers are here, exciting possiblities beckon. To do nothing is unthinkable. This paper reports one school system's response.

The Scarsdale Project

Two years ago students and teachers in Scarsdale, NY, were doing little with computers. Three or four terminals in an old office in the High School offered computer games to a handful of computer freaks, the terminals were down as often as they were up, and nobody seemed to understand what the freaks were talking about. Furthermore, inflation and declining enrollment had taken their toll in the school system: summer school and driver education had been abandoned, and other programs were threatened. The time hardly seemed ripe for a brave new curriculum venture in computing or anything else.

However the community does enjoy an enlightened citizenry and teaching faculty. Many were aware, however dimly, that computers were transforming the society and that they ought to be transforming the schools as well. Accordingly, in early 1978 the Scarsdale Board of Education appointed a citizen's advisory committee to make recommendations concerning the use of computers and computing in the schools. The committee's report, which gave the Board a platform for action, recommended that computing be introduced into the curriculum throughout the school system and that appropriate steps be taken to purchase necessary equipment and to train teachers in its use. The committee also recommended that the district engage a qualified consultant to help in the effort. Work began immediately.

We believed the project should begin with the education of teachers --as many of them, system-wide, as possible. Too many school ventures in the past had foundered because more money had been spent on expensive equipment than in helping people prepare themselves to use it. However, district history suggested that simply requiring teachers to take a new in-service course would not accomplish everything. Grass-roots support, from both teaching staff and community, would also be essential if the project were to succeed.

1978/1979

In the fall the District convened a steering committee consisting of about a dozen teachers and principals. Committee members were charged with keeping their colleagues informed of developments, reviewing progress, drawing up statements of assumptions and goals, and making recommendations for the purchase of more equipment when appropriate. With the consultant, this committee reviewed the Advisory Committee report, made plans for the first in-service education efforts and decided on immediate hardware purchase.

The committee swiftly acted to acquire hardware, both to have it available for the teacher training and to satisfy some long-standing student demands in the high school. However, the committee decided to limit first-year acquisition to a minimum in the belief that they could better make such decisions after they themselves had had a year of training and experience with computers. Because it was not clear that any one microcomputer would be best for everything, a mixture was acquired: four Apples, five Pets, and one TRS 80. As anticipated, the variety provided good experience in making later acquisition decisions.

As winter approached, Dr. Taylor met with all teachers in the school system to provide an overview of contemporary computer technology and its implications for teaching and learning. (The teachers actually met in several groups rather than in a single block: English and Foreign Language, Social Studies, Art, and Music; Math and Sciences;and Elementary. The purpose of these overview meetings was to inform teachers about the long range prospects and to build their enthusiasm for beginning training. All teachers were then invited to participate in the in-service course scheduled for 8 two-hour sessions throughout the winter months.

There were three main objectives to this first course: (1) to introduce the teachers to the rudimentary concepts of programming, (2) to get them over their fear and accustomed to using microcomputers, and (3) to better inform them of some potential uses of computers in the classroom. First, FPL, an language developed at Teachers College to teach programming, and BASIC were used to teach the crucial elements of programming. Second, scheduled use of the ten microprocessors was built into the course so that every teacher participating had to use them to finish the course. Third, selected articles by pioneers of computing and education were read and briefly discussed.

Formally, the course was offered jointly by the district and by Teachers College, Columbia University. The participants could take the course for Teachers College credit if they paid tuition, for local school credit if they paid the nominal teachers institute fee, or for no credit, without charge. Of a faculty of 330, 122 enrolled, including several principals and the superintendent. By the end of this course in late spring, one third of the Scarsdale faculty had at least some familiarity with computer programming and some hands on experience with a common microprocessor.

Meanwhile, meetings were held with parent-teacher groups and with members of the local press to explain the purposes of the project and to develop community support. When time came to prepare a new school budget, the Board of Education had no trouble in increasing its appropriation for computer equipment and additional training.

Determining the next steps was less easy. It is one thing to provide some teachers with an exciting introduction to computing, quite another to modify an entire curriculum by integrating computing into it. The vision shared by the authors and the steering committee was only general: computing should be incorporated into the curriculum. The problem was to refine this general goal into specific, detailed sub-goals. This task was all the more difficult because most members of the steering committee admittedly were not experienced in current computing technology let alone expert enough to predict what computers would be like two or more years into the plan when the curriculum changes might be realized. Nevertheless, planning proceeded.

To clarify what was underway, three documents (Appendices A, B, and C) were produced: a statement of assumptions, a set of curriculum goals, and a rough plan for managing project activity over a four-year period. Though almost immediately outdated by experience and events, these documents and the thinking behind them were essential foundations for further thinking and for much of the action that made them obsolete.

1979/1980

During the summer of 1979 a small group of teachers was paid to develop programming exercises for fifth and sixth grade children. (At this writing, at mid-year, many pupils have already raced through this material, despite the fact that till quite recently it might have been considered strictly high-school level material.) Based on the experience gained with using the first, mixed batch of microcomputers, the district selected and acquired 19 more: 15 purchased from school funds, two donated by an outside agency, and two purchased by parent groups. Each school then had a minimum of at least two machines. More were placed in the junior and senior high school and full-time aides were hired there to staff a computing center in each.

Throughout the school year the Scarsdale Teachers Institute (the in-service education teacher-run, teacher-serving program which jointly sponsored the first formal introductory computing course) has offered a series of mini-courses in programming in BASIC;

certain junior and senior high school teachers (primarily in math and science) have begun exploratory use of computers in their classes; a concerted effort has been made to introduce computing and programming to all fifth and sixth grade pupils in their classes; a selected group of K-2 teachers have begun to explore dynamic graphics applications in beginning reading and mathematics in their classes; and formal in-service work of several kinds has been continuing. At the same time, planning for subsequent years continued on various fronts.

Results

What has been accomplished thus far? To begin with, nearly one third of a district faculty which two years ago knew little or nothing about computers now knows the rudiments of programming, has lost most of its fear, and is beginning to try new things. A small cadre of teachers has become enthusiastic and increasingly knowledgeable and gives promise of leadership in the years ahead. Many pupils from the elementary grades through the high school have acquired a beginning knowledge of the computer's capabilities and of their own capacity to get the computer to help them think. Computer hardware exists in all the district's schools, with more to follow. And there is a broadening base of understanding and support of the project throughout the community.

Problems encountered

Any innovative project encounters inertia after the first enthusiastic push. Scarsdale now faces the problem of what to do about the two thirds of the faculty who have not learned about computers -- and what to do about the hundreds of enthusiastic children who have learned about them but must, perforce, be assigned to the classes of teachers who have not. There is the problem of providing support to the willing teacher who is learning but who needs help. There is the ubiquitous problem of money and of acquiring hardware as rapidly as it can be used.

And there is one other, more subtle problem that must perhaps beset all such ventures in computer education until the shape of the future is more clearly revealed. Computers have the power to change the ways in which people acquire and extend knowledge, just as writing changed such ways millenia ago. But until we know better how those ways will change and how a school should be reorganized to capitalize upon those new ways, computer technology is in danger of simply being harnessed for the pursuit of present goals within present modes of operation. To employ computers thus is to miss much of the revolutionary potential they represent and to heavily damp the enthusiasm and insight they might otherwise foster in children. To finish by using computers for little more than enhancement within the traditional curriculum and the presuppositions implicit in it would be tragic. The challenge Scarsdale and every other school or school district faces is to be sure this doesn't happen.

The Goal

Using computing to practice mathematics and language skills in traditional curricula is helpful but primitive, almost like using televison primarily to display pages from textbooks. Serious questions must be raised about mere productivity of educational approaches which limit visions of computer application to such narrow horizons. Some of the skills for which computer assistance is so easily designed may actually be of significantly less importance as computer access continues to increase. Hampered by ignorance, enticed by expectation, Scarsdale seeks a grander goal: to become so comfortable with computing that our understanding of it will naturally reshape the way we think about everything.

Concluding advice to other school districts

What might others learn from this project's successes and problems? We suggest the following:

1) Begin with people, not equipment. The eager person who wants the machine and is ready to employ it is more valuable and catalytic than the machine that no one understands.

2) Develop broad support throughout your community and teaching staff.

3) Engage in on going, system-wide planning. Do not leave matters to an individual school or department or to an outside consultant.

4) Use a consultant or consultants who can bring not only technical knowledge of computers, but an understanding of schools and the way people use their minds and imaginations.

5) Concentrate resources of time, money, and attention on the project. In this way you can achieve a sense

of purpose and direction despite the shrinking you may be suffering elsewhere in the system.

6) Don't wait for the perfect computer; current machines despite their costs and limitations are a perfectly good introductory device for teacher training and pupil experimentation. A great deal must be learned now if better machines are to be well employed when they do become available.

7) Don't limit yourself to one type of machine; get experience with several. As cheaper, more powerful ones bow in, you will have a better experiential background by which to judge them.

8) Find out what you can do with the hardware as it stands, then do it. Don't always wait for expert software; both teachers and children can learn powerful things from exploring a machine's basic capabilities to plot, edit, speak, delete, and so forth.

Appendix A

Assumptions re computers and computing in instruction

1) The influence of computing on human life and on the entire planet's development will continue to increase exponentially for the foreseeable future.

2) All pupils should be educated to cope with this growing influence; as many pupils as possible should be educated to master computing and to use it creatively.

3) Such educating toward computing should be general rather than specfic since hardware and software changes will continue to occur at such a rapid rate that specific training will rapidly become obsolete.

4) Any human being of normal intelligence can understand the fundamental principles of computers and computing, can operate computers, and can learn to write simple computer programs.

5) Pupils should begin learning about computers and computing from the time they first enter school and should continue such learning

throughout the grades.

6) Pupils should also use computers as an aid to other studies, in the humanities and social sciences and the fine arts, as well as in mathematics and natural science.

7) Pupils learn to use computers by using computers. The use of computers should be as thorougly integrated into the school program as the use of pencils or chalk.

8) Computers are both an object and an instrument of learning. Traditionally we study writing to learn how to write, and we use writing to refine thought and express feeling. So too should we study computers to learn how to use them and use computers to extend and refine our thinking.

9) If pupils are to learn about computers and computing, their teachers must learn about computers and computing. If all pupils are to learn about computers and computing, all teachers must learn about computers and computing.

10) Computer hardware will become increasingly less expensive, and both hardware and software will become increasingly available.

11) Computers can serve as tutors, teaching us skills and information according to programs written by others. Computers can also serve as tools, analyzing data, performing calculations, and so forth. And they can serve as tutuees as we teach them to solve problems, compose music, etc. Each use has its place, but the most important is the last.

12) Programming computers requires logic and precision. Pupils who learn to program computers practice logic and precision.

13) The study of computers and computing does not run counter to the spirit of humane studies and the exercise of free, creative intelligence. On the contrary, it extends and deepens them by extending the power of the human mind and may foster a much needed dimension to our intelligence.

Appendix B

Curriculum

1. From kindergarten through high school, at increasing levels of sophistication, pupils should learn the fundamental principles of computers and computing. Some pupils should learn to write simple computer programs in elementary school; all pupils should learn to write simple computer programs before the end of their junior high school years. In grades 9 or 10 all pupils should complete a one-semester course in computing, computer science, and the social issues raised by computer technology. Other high school computer studies should be oriented towards computer skills which students will need in college, such as computer applications in numerical analysis and statistics.

2. From kindergarten through high school pupils should use increasingly sophisticated calculating and computing devices as an aid in studying mathematics, science, and other subjects.

3. Throughout the grades interested pupils should have opportunities to extend and enrich their knowledge of computer programming. In the elementary schols these opportunities might take the form of after-school workshops or special individual or group projects; in the junior high school, of projects, mini-courses, or club activities; in the high school, of specialized elective courses, projects, or club activities.
   (How can we make this item more specific?)

4. Interested high school pupils should also have the opportunity to study computer science, computer technology, and data processing.

Appendix C

Curriculum: Four-Year Plan

1978-79

Elementary: None.

Junior High: Computer club plus?

High School: Computer Center open. Computers used in science courses.

1st semester: Introductory Computer Course.

Computers & mathematical application course 442 - 1/2 credit, 1 semester course.

Computer used in 9th grade math

Computer Club makes much use of center.

1979-80

Elementary:

1. Three week mini-course for intermediate students rotates among five elementary schools.
2. Consultant and 2-4 primary teachers work with K-3 students.

Junior High:

a. Each 8th grader has an equivalent of one week mini-course on computers (hands-on).

b. Each 8th grader has two week computer/literacy course in spring as part of social studies curriculum.

High School:

Introductory programming (2 semesters).

Advanced programming courses developed.

Computers used in science courses.

Computer applications in math developed.

Independent study projects available.

Computer Club active.

1980-81

Elementary:

Mini-courses for 5th/6th graders; most have hands-on time with desk-tops during both years. Increased/experimental activity at primary grades. Units for primary grades included in social studies, science curriculum guides.

Junior High:

Units on computers in social studies technology units.

Computer units in math courses both years.

Active Computer Club.

High School:

All before continues.

Increased computer use in social studies, business, and science courses.

1981-82

Elementary:

Mini-courses for all 5 -6 graders, all have opportunity for hands-on computer work.

Activities for primary grades included in social studies, science, and math guides.

Junior High:

Continuation of 1980-81.

High School:

Continuation of 1980-81. Expanded use of computers in non-math departments.

Staff Development: Four Year Plan

1978-79

First Semester.

a. Consultant meets with staff in various formats.

b. Guest speakers meet with staff.

c. Committee reads, discusses, meets, and argues. Engages others not on committee in similar conversations.

Second Semester.

a. Consultant continues to meet with staff.

b. Consultant teaches literacy course in Scarsdale.

Faculty visits other schools to examine use of computers.

Issues: What do we want kids to know:

What learnings are appropriate for each age?

How do we best teach about computers?

How do we go about organizing ourselves?

How do we involve the entire staff?

1979-80

a. Consultant advises computer committee and others.

b. Advanced seminar for those who had first course.

c. Four session course on BASIC for novices and literacy graduates (Sept./Oct.).

d. Follow-up mini-courses on educational applications & programming led by in-house staff (Nov. - February).

e. Course on graphics and music.

f. Guest speakers - Computer literacy.

g. Visits to other schools on an expanded basis.

h. Social studies, science, and business faculty gain experience with simulation materials through BOCES course.

i. Send one or two to computer repair school.

j. Course on programming for CAI (2nd semester).

1980-81

a. Consultant continues support group on computers.

b. One or two faculty take advanced college courses.

c. Course in FORTRAN, etc., offered to all with previous experience.

d. Introductory BASIC course repeated for those with no computer experience.

e. Release-time workshops for those with no computer experience.

f. LOGO on the APPLE course for elementary school teachers.

g. CAI programming course.

TASKS

1978-79

1. Prepare statement of aspirations.
2. Educate ourselves.
3. Design structure to involve entire district.
4. Plan and implement system-wide in-service program.

5. Plan 1979-80 computer budget (16,000)
6. Purchase 1979-80 computers and software.
7. Design courses at high school level.
8. Plan 3-4 yr. purchasing program for computers.
9. Design a mini-course for elementary students.

1979-80

1. Continue system-wide general education.
2. Provide follow-through in-service for teachers.
3. Plan 1980-81 budget (21,000 for hardware, software).
4. Make 1980-81 purchases.
5. Design courses at high school level.
6. Pilot units at junior high level during 2nd semester.
7. Pilot mini-courses at elementary school upper grade level. Experiment at primary level; expand primary level computer use, teach technology unit.
8. Develop maintenance procedures.
9. Integrate computers into expanded 5th grade technology unit.
10. Integrate calculators, computers into math curriculum.
11. Search for and write grant for computers.
12. Examine software.

1980-81

1. Focus in-service on advanced courses.
2. Continued expansion of curriculum offerings.
3. Plan 1981-82 computer budget.
4. Make 1981-82 purchases.

1981-82

1. High school curriculum in place.
2. Junior high curriculum in place.
3. Elementary curriculum in place.
4. Write computer scope and sequence.

INTEGRATING COMPUTING INTO K-12 CURRICULUM

Beverly Hunter
Human Resources Research Organization
300 North Washington Street
Alexandria, Virginia 22314
(703) 549-3611

ABSTRACT

Historically, schools and school districts have employed a wide variety of strategies to integrate computer-related activities into the curriculum. Several such strategies are reviewed, based upon a national sample. The need for a comprehensive computer literacy curriculum, K-12, is described. A plan to infuse computer literacy objectives and activities into the traditional curriculum is set forth. Implications of such a plan on teacher training and curricular materials are discussed.

Catherine E. Morgan
Dept. of Instructional Planning and Development
Montgomery County Public Schools
850 Hungerford Drive
Rockville, Maryland 20850
(301) 279-3321

ABSTRACT

Computer-based instruction in the Montgomery County Public Schools, a large public school system, includes computer-assisted and computer-managed instruction, computer literacy, computer mathematics, and problem solving. The interactive use of computers for children begins as early as third grade while the management system in mathematics monitors individual student progress from kindergarten.

Invited Session

FUNDING ACADEMIC COMPUTING PROGRAMS

Sheldon P. Gordon
Suffolk County Community College
Selden, New York 11784

Lawrence Oliver
National Science Foundation
Washington, D.C. 20550

ABSTRACT

At most institutions, the single most critical problem in the development, implementation, maintenance, or expansion of any academic computing program is lack of money. This presentation will focus on a variety of possible solutions to this problem. A number of alternate sources of funding, including grants from private foundations, grants from government agencies, and contributions from local business and industry, will be discussed. Primary attention will be devoted to the most likely source of outside funding, namely government grants, especially those available through the National Science Foundation. In particular, details will be provided about the respective objectives and limitations on those NSF programs that would most likely sponsor various types of computer oriented activities. These would include CAUSE (Comprehensive Assistance to Undergraduate Science Education), LOCI (Local Course Improvement), ISEP (Instructional Scientific Equipment Program), and MISIP (Minority Institution Science Improvement Program).

In addition, the presentation will include a discussion of some of the important do's and don'ts connected with proposal writing, and will relate them to the process of grant review as conducted by the NSF.

180

Tutorial

TECHNIQUES FOR INSTRUCTIONAL SOFTWARE DEVELOPMENT USING MICROCOMUTERS

Kevin Hausmann
Minnesota Educational Computing Consortium
2520 Broadway Drive
St. Paul, Minnesota 55113

ABSTRACT

As microcomputers become more and more available and easy to use, more and more people will be designing and writing software for them. Before a project is begun, however, there are several obvious but often gotten considerations:

1) Is this a reasonable application for a microcomputer?
2) What are the limitations of my equipment?
3) What modes of interaction should be used?
4) What support materials may be required?

Once the analysis phase is completed, the design and layout stage may begin. Design should center primarily on the organizational content and division of the material. Also at the design stage, one should consider the mode of presentation of the material. So far, all the time spent on the project has not involved any time programming the microcomputer.

After the design is planned out, one enters the implementation stage. Briefly, some items to consider here include:

1) Adequate spacing of material on the screen.
2) Efficient movement between frames.
3) Effective presentation techniques.
4) Good ways to ask questions.
5) Effective feedback for answers.
6) Adequate and appropriate use of graphics and other special features of the microcomputer.

After an application is developed and written, it is very important to adequately test the program. Many times this is best done by watching someone use the program who is totally unfamiliar with it.

More detailed written material of this nature is available from the Minnesota Educational Computing Consortium, 2520 Broadway Drive, St. Paul, Minnesota 55113. Ask for the Apple Authoring Guidelines.

Mathematics

A METHOD FOR EXPERIMENTING WITH CALCULUS USING COMPUTER-ASSISTED INSTRUCTION

Frank D. Anger & Rita V. Rodríguez
Department of Mathematics
University of Puerto Rico
Río Piedras, Puerto Rico 00931
(809) 764-0000

INTRODUCTION

The project which is the object of this report was carried out in the Faculty of Natural Science at the University of Puerto Rico, Río Piedras campus, during the years 1976 to 1979. The National Science Foundation sponsored the project through its Minority Institutions Science Improvement Program (MISIP), making possible a whole new aspect of instruction and learning for 1600 students. Although the present paper will concentrate on the computer-assisted instruction modules developed for the introductory calculus course that we believe are innovative themselves, we must begin by describing briefly a few points which make this project, as a whole, unique.

First of all, as the composition of the student body is that of the Spanish-speaking culture of Puerto Rico, the various materials of the project were produced in Spanish. In addition, experience has shown that although UPR mantains a selective admissions policy, entering students are frequently deficient in analytic skills and in experience with methods of scientific inquiry. Inadequate or, frequently, incorrect preparation may lead a student to frustration or failure when faced with laboratory or problem-solving situations.

Second, the project introduced or advanced sweeping changes in the curriculum. Rather than simply paste on a little computer-assisted instruction (CAI), we took into account the full importance of the computer to modern science and in particular to the working scientist. Thus, programming was made mandatory for all majors in the faculty, CAI was introduced in all basic science courses, and interactive computer utilities were made available for students of certain advanced laboratories. In the calculus courses, greater emphasis was placed on the use of the calculator and on the numerical methods of calculus.

Finally, the project created the Natural Science Academic Computer Center headed by a member of the faculty and staffed exclusively by students. Rather than implanting a ready-made system to which students have recourse, we thereby created an organic extension of the students' environment, providing them immediate incentives for probing deeper into the uses and operation of computers. This last area has been one of the high points of the project. (4)

THE MATHEMATICS COMPONENT

The project developed in an orderly series of stages, beginning with the training of staff, the search for appropriate hardware, and various necessary administrative measures. Although every part of the project seemed to be fraught with unexpected difficulties, the faculty as a whole responded more positively than expected, and their cooperation along with that of various student assistants kept things moving forward at all times. In the Mathematics Department, the necessary changes and investment of time were the greatest, since to this department fell the full weight of training professors and implementing the new programming course requirement for the whole faculty. We soon realized that the applications in the different disciplines involved were better handled by two new programming courses rather than by the old course (see Figures 1.a and 1.b). The pre-medical and biological science students were better served with a programming course that would include data gathering, statistics, and examples in their own discipline. There are many packaged programs that, with some programming knowledge, the student may easily adapt to his needs. On the other hand, the physics, chemistry, and mathematics majors were to be taught more sophisticated programming, also with examples from these disciplines.

To train more of the faculty members in PL/1, programming seminars were given. We chose the PL/1 language in particular because we believe that it is extensive enough that the student who knows it may teach himself whatever language the machine he has at hand uses. Parallel to this, the calculus course was divided into two courses: the first, as in the programming course, designed to serve the pre-medical and biological science students and the second to serve physics, chemistry, and mathematics majors (see Fig. 1.a and 1.b). These divisions respond to long-standing curricular tensions in a faculty in which about two thirds of the students will go no further in mathematics than the first semester of calculus while the other third will need at least two or three semesters of more advanced mathematics.

THE COMPUTER-ASSISTED INSTRUCTION FOR CALCULUS

The major reason for the implementation of CAI at the University of Puerto Rico was to combat a lack of analytical skills and to overcome, to whatever extent possible in such a limited program, the tendency toward rote learning and memorization as the basis of learning. We desired, moreover, to avoid doing with the computer what could be done equally well or better by more traditional methods, with the possible exception of the one-to-one tutorial. Thus, we sought three basic things from each module:

1. It should perform some graphic or numerical simulation of some fundamental process, concept, or technique.
2. It should be highly interactive and open ended, allowing the student some freedom to experiment with the concept under investigation.
3. It should minimize the difficulties faced by the student using the computer.

These objectives have much more obvious implementations in the experimental sciences than they have in mathematics, and we feel that there is still much room for exploration and development in this direction. Nonetheless it is herein that we feel much of the novelty and value of our modules lies.

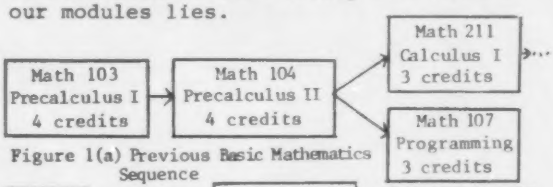


Figure 1(a) Previous Basic Mathematics Sequence

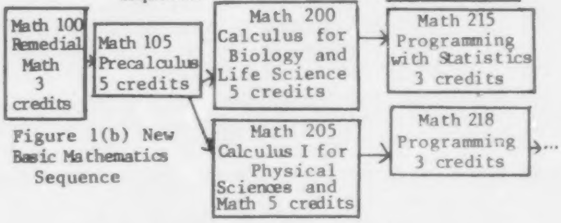


Figure 1(b) New Basic Mathematics Sequence

Before the actual structure and content of the individual modules are discussed, it will be instructive to look at some of the story of their development. At the start of the project we wrote to numerous publishers and universities in order to obtain information on texts and already existing CAI for calculus couses. Although we then found some interesting supplementary texts, it soon became apparent that very little had been done for calculus like the interactive programs that we envisioned. Algebra, statistics, and linear algebra seemed to be the areas that had attracted the most effort for a variety of reasons running from the size of the student populace to the appropriateness of the material for programming. We found the nearest thing to our aspirations at Georgia Institute of Technology where Dr. J. C. Currie and a small group had developed some rather brief programs for illustrating limits, derivatives, approximate areas, and other topics along with a driver program which we totally failed to appreciate at the time. (6) Far more impressive things had been done for logic at Ohio State University, for statistics at University of Akron, and for non-mathematical subjects in many other places; we wanted to see something similar done for calculus.

As we learned more about the difficulties of compatibility and transferability and thought more about the problems of translating whatever we found into Spanish for our students, we leaned towards developing our own programs out of the many materials and ideas which we had collected during the first year of the project. We had finally settled on buying a Hewlett-Packard 2000 System for its reputation for fast response with a large number of users, to give us independence from the main computer center with its many large administrative jobs, and because it supported Coursewriting Facility, an approximate implementation of IBM's Coursewriter with available BASIC numerical functions. We had experimented with the latter language on UPR's IBM 370/145 computer and believed that it was a rather powerful tool providing the necessary structure for developing good CAI. We were eventually to come to grief with the Coursewriting Facility, but we remain pleased with the rest of our system; after many months of work in this language we found ourselves unwillingly forced to convert everything to BASIC, which is now the language in which all our programs are written.

We arrive, then, at the modules themselves. It was for the calculus course of the hard sciences and mathematics (Math. 205 in Fig. 1) that the CAI modules were produced. The various programs which make up the entire CAI system in calculus are depicted in Fig. 2.

Within the system, the student has complete freedom to choose among the different modules at any time, although he is informed in class when it would be most appropriate to study each module. The manager program, which interfaces between the student and the individual lessons, also allows the students to register themselves, avoiding one of the time consuming operations often associated with the beginning of the semester. The structure of this program is shown in Fig. 3. (Of course the security with this system is minimal, but in our environment this is not a problem.) This same program maintains data on the total number of uses of each module by each student and the total number of minutes of terminal time of each student. Three separate report programs have been written to produce different usage reports for the teachers of each section and for global evaluation. No data are kept on right and wrong responses: it will become clear further on that such information is either not applicable or irrelevant for the majority of response situations presented.

The general form which most of the modules follow is given in Fig. 4. It must be remarked here that these modules are not in any way conceived of as replacing the textbook or the lecture. They are strictly supplementary and each one presupposes that the student has already been introduced to the concept or technique in his regular class work. Thus, in the first part of the module there are usually a few questions to find out if the student is at all familiar with the material, and if not, to recommend that he learn more about it before going on. The examples which the computer presents are as much for the sake of familiarizing the student with how the computer output looks and how to later enter his own data as they are for illustrating the material being studied. That is to say, it is expected that the student will continue making up his own examples and investigating the computer's responses and will do most of his learning during this latter exchange. Those students who stop after the prepared examples are not really likely to have gained much. Fig. 5 presents the eight modules and their content essentially as the manager program presents that menu to the students, except of course, it appears in Spanish on the screen.

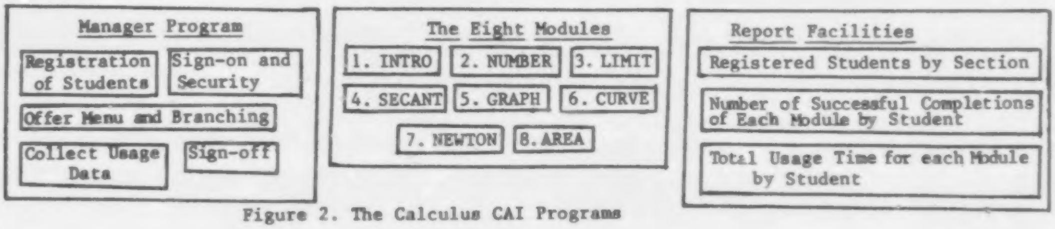


Figure 2. The Calculus CAI Programs

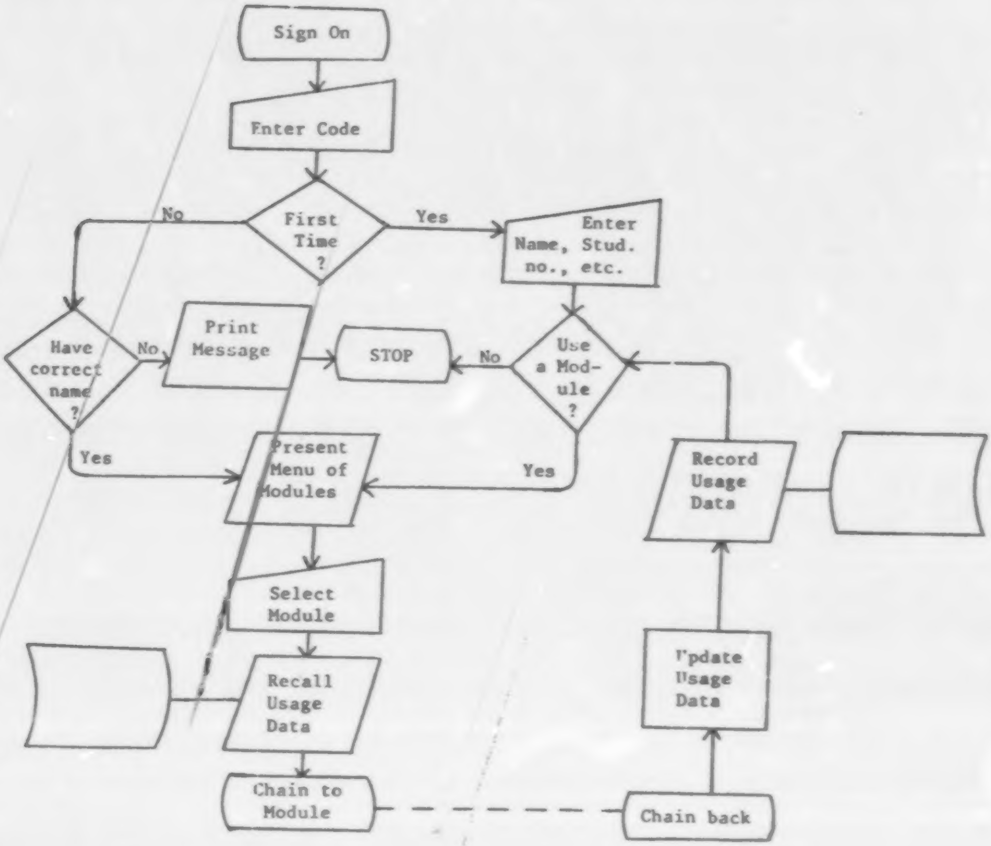

Sign On
Enter Code
First Time ?
No
Yes
Enter Name, Stud. no., etc.
Have correct name ?
No
Print Message
STOP
No
Use a Module ?
Yes
Present Menu of Modules
Yes
Record Usage Data
Select Module
Recall Usage Data
Update Usage Data
Chain to Module
Chain back

Figure 3. Manager Program Flowchart

1. INTRODUCTION
2. QUESTIONS. Does the student know enough to benefit from the module? If not, he may be asked to return to the text before continuing.
3. INTERACTIVE EXAMPLES. The student participates in working through an example selected by the program.
4. EXPERIMENTATION. The student is given the opportunity to make up and enter his own examples in the same format as in 3 . The computer provides the necessary data and prompting. The option to terminate is offered before each example.

Figure 4. General Scheme for Modules

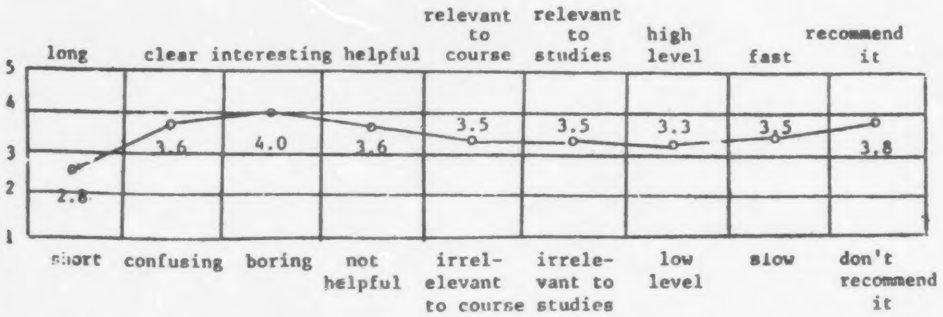
1. INTRO :	Introduces the terminal, the keyboard, and its traits.
2. NUMBER:	Presents real numbers, round-off, and how to read mathematical tables.
3. LIMIT :	Studies limits of rational functions with values that make the denominator equal to zero (0) by the use of a table.
4. SECANT:	Illustrates how the slope of the secant tends toward the slope of the tangent.
5. GRAF :	Calculates values of a given function and its derivatives in order to help plot the graph.
6. CURVE :	Evaluates the function, finds the intercepts, critical points, and points of inflection of a polinomial.
7. NEWTON:	Finds the roots of a given polinomial using Newton's Method.
8. AREA :	Calculates the approximate integral by various methods.

FIGURE 5. The Eight Calculus CAI Modules

Table 1. Calculus Module Usage

	Number of Sections	Total Students	Number of Students participating	% of Students participating	Modules Used per Student	Average Total Minutes
Fall 78-79	5	150	37	25 %	3.3	65
Spring 78-79	9	270	123	46 %	5.6	84
Fall 79-80	7	200	67	34 %	3.1	68

Table 2. Average of Responses to Module Questionnaire, 1978-79



The first two modules in fact have nothing to do with calculus. They are there to minimize the shock of dealing with the computer for the first time and increase the probability that the student will be able to successfully use the modules and interpret the results. Next comes a module on limits and then on the derivative, both centered around investigating tables of values of the appropriate expressions in the neighborhood of a chosen point. There are no epsilons or deltas here, and it is hoped that the student will get more of a feel for how varying values approach a fixed number. The next two modules deal with the use of the first and second derivatives to describe certain properties of the graphs of functions and to draw those graphs. It is our continuing disappointment that these modules contain very few graphs. Despite the fact that we had proposed to buy some graphics terminals and even a plotter, we were never able to do so. Several programs were developed to produce graphs with alphabetic characters but were never actually incorporated into the final modules. It is still not clear whether a graphing routine cannot make a straight line look like a straight line that will be very convincing for the students. The seventh module illustrates the Newton-Raphson Method for finding approximate roots to polynomials, while the last module does approximate integration by three of the standard methods frequently treated in introductory calculus courses.

To demonstrate the implementation of the general structure of the modules as shown in Fig. 4, we shall discuss in more detail the seventh module: Newton. The introduction reminds the student of the objective and iterative nature of Newton's Method and requires him to recall the precise form of the iterative formula, using one of the very few multiple choice formats in these modules. Once he gets that straight, the sample function, $f(x) = x^3-5$, is presented. The student is directed to choose a starting value, x_0, and the computer then calculates, one at a time on request, x_1, x_2,... until the student decides that he is ready to say what is root being approximated. Upon entering his guess, he is either told it is not good enough and to look at more values, or that it is right, or that it is close enough, but that a better value would be...Once the student has successfully named the root, the computer offers him the opportunity to enter a polynomial of his choice by entering, successively, the degree and the coefficients.
The program then runs through the same sequence of steps until the student names the root being approximated with sufficient accuracy or until it becomes clear that the process is not converging. (As you may expect, the only method the program has for determining the root is the same as Newton's Method, so that in some untidy cases the computer gets it wrong, leading perhaps to some confusion. This sort of thing is, however, a constant source of difficulty in many of the modules and in fact in all numerical methods. Some care has been taken in the programming to avoid some of these pitfalls and to give some warnings in the text portions. The student manual, which offers various forms of encouragement and hints, also points out some of the difficulties.) The student then has the option of trying for another root of the same polynomial, entering another polynomial, or terminating the module.

RESULTS OF THE CAI IN CALCULUS

Throughout the duration of the project, and continuing now, certain evaluative procedures were used to assure some kind of objective information about participation, reaction, and effect of the CAI. The first of these, participation, was constantly monitored by the computer itself. Table 1 shows the percentage of the enrolled students who actually used the modules to some degree or other, the average number of modules run by each student, and the average time spent by each student at the terminal during the semester. The relatively low use is due principally to three factors. Foremost is the totally voluntary nature of the computer laboratory. Any such voluntary activity cannot hope to attract more than 75% of a class unless it guarantees better grades. Our modules offer no such guarantee. Secondly, there were considerable difficulties during the first semester of 1978-79 due to the delay in completing the final home for the computer center and the resulting lack of terminals and comfortable working conditions. At that time there also was no student manual, leaving the students more or less on their own in attacking the modules. Finally, the professors teaching the various sections of the calculus course change from semester to semester, and there is a clear correlation between the enthusiasm of the professor for the computer laboratory and the students' participation.

Student reaction was checked each semester by a standard questionnaire distributed in class and in the computer center. It was originally supposed that a student would fill out one of these questionnaires for each module that he used, but we soon decided that we should be content if we could elicit one overall reaction from each participating student. The results of these questionnaires are shown in Table 2. The reaction is clearly favorable, being most favorable on the issues of "interesting" and "would recommend it." (Responses here were made on a scale from 1 to 5, 1 in full agreement with the bottom description and 5 with that of the top.) The form used also included more direct questions over problems encountered in using the modules, and some of the students' comments led to corrections and minor improvements.

The actual effect of the CAI on student performance and general understanding is obviously the most difficult to measure of the three parameters, but perhaps the most important. We have made two formal attempts at measuring this. The first of these was in the second semester of 1977-78, when we set up non-participating control sections for comparison of results. Due to a plague of difficulties, many relating to the problems which led to giving up the Coursewriting Facility, it became impossible to gather any reasonable data. The second attempt was made at the end of the first semester of 1978-79. At that time a study was made comparing certain indices of calculus students who had actually used some of modules. These indices were their SAT scores, both achievement and aptitude, their final grade in calculus, and the number of modules used. In this small sample, no actual statistical correlation could be established, but certain tendencies were apparent. For example (see Table 3), all the students who used five or more modules obtained a grade of C or better in the course, while of the students who used five or more modules, only one had obtained better than 700 on the SAT math achievement. (One fourth of the participating students ran five or more modules). One is tempted to draw the conclusion that the students who used the modules most are the hard-working students; not necessarily highly intelligent, but nonetheless successful.

Table 3. Comparison of Module Usage Course Grade, and Previous Aptitude and Achievement Test Results Fall Semester 1978-79

(Figures in percent of participating students.)

Achievement Scores	1-2	3-4	5-6	7-8
701-800	27	4	4	
601-700	11	8		8
0-600	11	15	11	

Course Grade	Aptitude Scores			Number of Modules			
	0-600	601-700	701-800	1-2	3-4	5-6	7-8
A		4	27	15	11	4	
B	8	4	8	4		11	4
C	4	4	11	8	8		4
D	4	8	4	11	4		
F	11	4		11			

To conclude this discussion of the evaluation of the project's results, we would like again to recall that the objectives of these CAI modules and the basic philosophy used in constructing them are to overcome certain experimental and analytical deficiencies in the students' preparation (as well as to offer a challenge to the more highly motivated students), and hence they are not directed at the routine type of problem solving that is prevalent on examinations. It was, therefore, never expected that using the modules would be directly and immediately reflected in better test scores, but rather in a deeper, long-range appreciation of fundamental concepts and a more enquiring approach to attacking new material. Any correlation obtained between exam performance in the calculus course and module use is therefore much more likely due to pre-existing attitudes of the student than to the contribution the calculus CAI made to his or her knowledge. The calculus is neither a bag of technical tricks, as it is frequently taught to non-science majors, nor an exercise in mathematical logic, as it often appears in honors courses. It is a coherent system of ideas elaborated for the modeling and analysis of certain classes of phenomena; experience with some of these phenomena and with the way calculus purports to capture these processes is necessary for anyone aspiring to scientific investigation or teaching. We hope that our modules make a start in the direction of providing some of this experience.

DIFFICULTIES AND WARNINGS

The creation and implementation of successful computer-assisted instruction is a major undertaking involving many factors, any one of which is capable of nullifying many months of serious effort. Our project, which we consider to have been reasonably successful overall, did not escape from its share of mistakes and frustrations, and it still faces the difficulties of maintenance and improvement. The initial challenge is the selection of an adequate system of hardware and software, unless one is locked into a pre-existing system. The choices today are far wider than they were three years ago, and serious consideration must be given to microcomputer systems as well as to minis and main frames. Although our mini computer, the Hewlett-Packard 2000 System, has functioned well in our environment, its upkeep may be too great for a smaller institution, or capacity too limited for a more ambitious project. The lack of special symbols on our terminals (H-P 2640B), such as integration signs or even lower-case letters, puts undue strains on programmers and users alike. The mathematical symbols as they appear in texts and are used by the teacher on the board are important for a quick understanding of the material and for adequate reinforcement. For example, INT X^2 DX needs much more attention than does the usual expression, $\int x^2dx$. Related to this point is the ability of the terminal to recall previous pages of material (terminal memory), or careful programming to imitate this ability. Although all our terminals now have memory, at the outset of the project they did not, and we found this situation much less flexible and more demanding.

Problems with software can be more insidious. It is not easy to find out from salesmen exactly what a computer or a software package is capable of doing, particularly in the area of educational applications. We began, as mentioned above, with IBM's Coursewriter which is an especially designed system for CAI and its management. We soon discovered, however, that it is incapable of doing any kind of calculations other than integer arithmetic. This can be overcome by begging, buying, or writing assembler-language functions to increase the power of Coursewriter, but this solution we found to be painful and restrictive. H-P, on the other hand, allows these functions to be added to their Coursewriting Facility via simple BASIC programs. To a large extent this is what sold us on the system we bought. Several months later we gave up trying with the Coursewriting Facility and converted to BASIC. Complications in the compiling procedure, bugs in the management facility, and system crashes occasioned, at best guess, by the many needed disk accesses finally wore down the patience of even the most stalwart among us. Of course, BASIC is an entirely adequate language for most CAI (consider its transferability, by far the majority of the available CAI packages are in BASIC), but it does not encourage the same kind of careful organization and answer processing as do Coursewriter and other CAI languages.

Some of the other components that can, and hence will, go wrong are inadequate physical plant, too few terminals, improper scheduling of student usage, poor response time when the computer is busy, and, inevitably, equipment failure. While it is idle to suppose that one can avoid all of the problems and delays that typically arise, careful consideration can avoid many of the difficulties. After all, you can not change your computer system with the same frequency and ease with which you have been changing your calculus text over the years! We hope that as the advice and experience of a large number of generous and helpful people helped guide us through the vagaries of this project, so some of our experience may contribute to future endeavors of this nature.

REFERENCES

1- M.J. Christensen, *MATHDOC: A Control System for Computer-Assisted Calculus*, Proceeding of Conference on Computers in Undergraduate Curricula, Denver, June 1978.

2- W.S. Dorn, G.G. Bitter, D.L. Hector, *Computer Applications for Calculus*, Boston: Prindle, Weber and Schmidt, Inc., 1972.

3- G. McCarty, *Calculator Calculus*, Palo Alto: Page-Ficklin Publishing Co., 1975.

4- R.G. Selsby and M. Gómez Rodríguez, "On-the-Job Training of Students in Computer Science," *Proceedings of Minority Institution Curriculum Exchange Conference*, Concord, North Carolina, January 1979.

5- D.A. Smith, *Interface: Calculus and the Computer*, Boston: Houghton Mifflin Company, 1976.

6- K.D. Stroyan, *Manual for a Computation Laboratory in Infinitesimal Calculus and Linear Algebra*, Mathematical Sciences, University of Iowa, 1976.

COMPUTER APPLICATIONS IN A FINITE MATHEMATICS COURSE

by

K. Abernethy,
G. Piegari,
and
A.L. Thorsen

Mathematics Department
Virginia Military Institute
703-463-6335

INTRODUCTION

The course described in this paper supplements a standard course in finite mathematics with computer applications. It was developed with the following goals in mind. First, since the course is offered to students in engineering or mathematics curricula, it was the authors' hope that the computer applications would increase student interest and motivation as well as head off the common complaints about the irrelevancy of abstract mathematics courses. Second, it was felt that the computer assignments would lead to an increase in the student's proficiency in programming, which would prove beneficial in his later mathematics and engineering courses. Finally, it was hoped that the student's grasp of certain topics, in particular probability, would be enhanced by the computer assignments.

Three sections of the course were taught to a total of 51 students in the spring semester of their freshman year. Each of the students had received, in the fall semester, at least 4 hours (contact hours, not credit or semester hours) of instruction in BASIC. However, most were far from being competent programmers, and the authors are convinced that our course could be made self-contained by beginning it with 4 or 5 lectures on BASIC. We started our course with a diagnostic test on BASIC and followed it with a one-day review of BASIC. We resolved to discuss programming only as specific problems arose in connection with computer assignments and spent no more than an average of 15 minutes class time per week on programming. Eight computer assignments, timed to correspond to the material being covered in class, were given. Each assignment was to be completed by its assigned deadline, which was between one and two weeks. These assignments, in total, were counted as an hour test (15% of the final grade in the course). No computer work was included on hour tests or the final examination.

The computer assignments were executed on an HP2000 computer through 8 time-sharing terminals in three locations on campus. However, a group this size could easily have been accommodated with as few as three terminals (perhaps fewer with proper scheduling). To provide help on the computer assignments, each of the three instructors was available for a one-hour scheduled help-session each week. Each session was open to all students in the three sections and was conducted in a computer terminal room where five terminals were available.

One restriction on the course, placed by our department, was that none of the core mathematics content of the course be deleted or diluted in order to accommodate the computer work. We believe that we satisfied this restriction, even though certain introductory topics from chapters 1 and 2 of the course text (1) were omitted from the syllabus to provide the additional time required for the computer assignments. The time gained proved more than adequate, and in addition to the computer work we were able to give a reasonable treatment of linear programming, which was not in the original course syllabus. The major topics in the course were set theory, discrete probability (including Markov Chains and stochastic processes), matrix algebra, systems of linear equations (including a Leontief model in economics), and linear programming. The text material was chapters 3,4,5, and 6 (1), but there are numerous textbooks which offer similar coverage.

The course was evaluated in two ways. A student evaluation form was completed by each student at the end of the course, which provided some insight into whether the goal of increased interest and motivation had been achieved. More formally, we compared the performance of the students in the computer-supplemented course to that of students in the standard course. A large part of the final examination which was given in our course had been given to several earlier sections and one concurrent section of the unsupplemented finite mathematics course. To test our hope that comprehension of some of the mathematical topics would be enhanced by the computer work, we compared the scores of the students in the standard course with the scores of the students in the computer-supplemented course for the relevant sections of the exam. Results of these evaluations are discussed in section four of this paper, following a discussion of some of the computer assignments in sections two and three.

COMPUTER ASSIGNMENTS IN PROBABILITY

The first five of the eight computer assignments were related primarily to probability theory. Four of these five were simulations of probability experiments. One assignment was a program to compute combinations and permutations and use these results, in turn, to compute the probabilities of certain events.

The first program was simulation of one hundred tosses, one thousand tosses, and then ten thousand tosses of a fair coin. It was in this assignment that the students were introduced to the random number generator and its vital importance in simulating experiments. Also, we indicated the need for a test condition to translate the value of the random number generated into an outcome for the experiment. For example, since the random number generated is greater than or equal to zero and less than one, declaring the outcome as heads if the number is less than 0.5 is a simple and reasonable way to simulate the tossing of a fair coin.

The second assignment used the random number generator introduced in assignment one for a more complicated simulation. The experiment was rolling a pair of unbiased dice. The student was reminded that the outcome of one die is independent of the outcome of the other, and the outcomes 2 through 12 must be generated in conformity with their natural frequencies. The use of the integer function INT was introduced. The outcome of one die can be simulated by generating a random number, multiplying it by six and adding one, and then taking the greatest integer (INT) of this result. The outcome of each die is generated independently, and the two numbers are added together to obtain the outcome of the experiment. Assignment two consisted of running 100, 1000, and then 10,000 trials of the experiment and approximating the correct probabilities of the various outcomes. Varying the number of trials was intended to convince the student that the larger the number of trials, the closer the approximating probabilities reflect the true probabilities obtained by analytical methods in the classroom.

The third computer program was a simulation of the classical Buffon's Needle Problem. It is both easy enough to simulate on the computer and difficult enough to defy solution by the analytical methods studied in finite mathematics (although it can be solved by integral calculus). The student is thus introduced to the main value of simulation--solving problems that cannot be solved by other means. In Buffon's problem, a needle one unit in length is dropped on an infinite grid of vertical lines. The probability that the needle touches a line is the desired quantity. For our experiment the lines were two units apart. In order to accurately simulate the experiment two independent random numbers must be generated, one to obtain the angle the needle makes with a horizontal line and one to obtain the location of the center of the needle between two lines. The test condition to determine a hit or miss amounted to comparing the projection of the needle on the horizontal line to the distances of the center of the needle from the adjacent vertical lines (5). The student was required to compute the approximate probability that the needle hits a line for 100, then 10,000 simulations of the experiment. Incidentally, since the exact probability is $1/\pi$, we asked the student to compute the reciprocal of his approximate probability to see if he would recognize this answer as an approximation of π.

Assignment four involved programming algorithms to compute C(n,r) (the number of combinations of n objects taken r at a time) and P(n,r) (the number of permutations of n objects taken r at a time) and using these algorithms to compute probabilities, for example the probability of getting four aces in a seven-card poker hand. One by-product of this assignment is the lesson that computers have practical limitations. The largest factorial that could be computed on our computer was 33!. Therefore, the student was forced to develop a more sophisticated algorithm to compute C(48,3)/C(52,7) (the probability of four aces in a seven-card hand) than simply computing each of the factorials first and then performing the division.

The fifth assignment had three parts: (1) the simulation of 3000 World Series between two evenly matched baseball teams; (2) the simulation of 3000 World Series in which one team had a .6 probability of winning any single game; and (3) the simulation of 3000 World Series if it were changed to a best of nine game series, rather than a best of seven. In the first part, where the teams are evenly matched, the length of each series was tabulated, to illustrate the non-intuitive fact that a six-game world series is as likely as a seven-game series. Since the probabilities of having a six-game series and a seven-games series are equal, about half the class can be expected to conclude that a seven-game series is most probable and half that a six-game series is most probable. It should be noted that a slight deviation away from exactly evenly matched teams makes a six-game series the most likely. The second part of the fifth assignment was intended to illustrate the fact that if one of the teams has an advantage over the other team in each of the games, then its advantage is magnified when the trials are grouped into a contest. The probability that a team with a .6 probability of winning each game wins the series is about .7. The third part of the assignment illustrated that as the number of trials grouped in a contest are increased, the initial advantage is further magnified, in this case from about .7 to about .73. Although each of these parts can be solved analytically, the solutions are complicated and usually beyond the objectives of a finite mathematics course; so, the student is again impressed with the primary value of simulation. (Similar problems are discussed in Kemeny (2), pages 143-144 and section 1.5 and Kemeny (3), pages 161-167.)

COMPUTER ASSIGNMENTS USING MATRICES

Assignments six through eight were designed to introduce the students to the matrix (MAT) commands in BASIC and to the system library. Assignment six was a set of routine exercises that required the use of the commands for the matrix operations of addition, multiplication, scalar multiplication, multiplicative inversion, and transposition. Additionally, the students were given a 2x2 matrix B and asked to use a loop to determine B^4.

Assignment seven paralleled our class discussion of Markov Chains, and its purpose was to introduce the students to subroutines, round-off error, and a double loop. For the project, the students were given two 3x3 transition matrices and asked to write a program to test each matrix for regularity. A transition matrix A is regular if A^n has all positive entries for some nonnegative integer n. The students were asked to recall that in performing calculations with a computer, round-off error is almost always present. Hence a quantity, which may actually be equal to zero, may be calculated by the computer to be, for example 0.0000021. Thus, in the subroutine to test for zero entries in $B=A^n$, the students were forced to structure their conditional statements in such a way as to ignore any difference in compared quantities which are less than the suspected round-off error introduced by the computer. In addition to a loop to calculate A^n, assignment seven also required a double loop (one for the row index and one for the column index) to determine if A^n had any zero entries. The transition matrices in the assignment were 3x3 matrices. Thus the students knew that their program need only test $A^1,\ldots A^5$ for zeroes because in class we gave the theorem stating that if an nxn transition matrix is regular, then at least one of the first $(n-1)^2+1$ powers of A contains no zero entries.

For the first part of the final computer assignment the students were required to write a program to solve two problems based on our classroom discussion of Leontief models. This program involved solving a system of linear equations using the INV function to find the inverse of the coefficient matrix.

It has been noted that one of our objectives was to increase the student's proficiency in programming. Toward this end, each of the above computer assignments required the students to actually write a program--not merely supply data to a program called from the system library. However, the authors felt that students should be given an opportunity to realize the importance of the system library; hence, they introduced the system library in the second part of assignment eight. The problem supplemented our class work on linear programming and the simplex method. In the assignment the students were asked to maximize an objective function subject to five constraints using a program (LINPRO) from our system library. The instructors felt that the introduction to this powerful facet of computer use was an appropriate end to the computer segment of the course.

EVALUATION OF THE COURSE

The results of the student evaluation given at the end of the course were generally quite positive. Most students seemed not only to enjoy the course, but to feel that it was useful. For example, a question on the overall rating of the course had an average of 8.3 on a 0-to-10 scale. The average response to the question: "Would you recommend this course to others?" (again on a 0-to-10 scale , where 0 = not recommend at all, and 10 = highly recommend) was 8.1. In addition, many responses in the comment section of the evaluation form indicated that the course was well-received.

A statistical evaluation of the course was done by comparing scores of students in the regular course (group R) on 21 problems common to both final exams. We proposed average scores on the mathematics portion of the SAT as a possible measure of how groups C and R compared in ability. Immediately a problem arose because the two groups were significantly different when this variable was measured. Two comparable groups were acquired by deleting all students with Math SAT scores over 600 in each group of students. We were left with 24 students in group C with an average Math SAT score of 531, and 29 students in group R with an average Math SAT score of 533. Students in both groups C and R were engineering and mathematics majors.

The results of the 21 examination questions are given below.

	Group C	Group R
Number of students	24	29
Average Math SAT	531	533
Total number of questions	504	609
Number of correct responses	330	365
Sample success probability	$\hat{P}_C = .655$	$\hat{P}_R = .599$

From the table above we get the difference of the sample proportions to be $\hat{P}_C - \hat{P}_R = .056$. Let P_C be the success probability for the group C population and P_R the success probability for the group R population. Testing the null hypothesis $P_C \leq P_R$ against the alternate hypothesis $P_C > P_R$, we found that the probability that $\hat{P}_C - \hat{P}_R > .056$ is less than .028 (c.f. chap. 8,(4)). Therefore, at the 5% level of significance, we rejected the null hypothesis and accepted the alternate hypothesis that the performance of students in the computer-supplemented course would exceed that of students in the regular course on the 21 exam questions. We point out that the one-sided test was suggested by the restriction that the coverage of the central mathematics topics in the new course be at least equal to the coverage given in the regular course. Consequently, it was expected that the value of P_C would be at least that of P_R.

There may be other factors which contributed to the superior performance of group C. For example, those in group C were exposed to each of the three instructors, while those in group R were taught by only one of these instructors. In addition, the class size for group C averaged 17 students, while the average class size of group R was 24 students. However, the authors believe that the introduction of the computer into the course was an important positive factor contributing to the significant difference in performance noted above.

Finally, we remark that when the students with Math SAT scores of over 600 were compared, the students from group C again performed better on the 21 exam questions than the students from group R. However, the difference was not as significant. This result is consistent with the authors' feeling in the beginning of the project that the better students were likely to perform well no matter how the course was designed and that the greatest effect of the computer supplements would be on the average students.

SUMMARY AND RECOMMENDATIONS

It was our intention to add a computer supplement to a course in finite mathematics without compromising the traditional approach to the mathematical content in the course. In short, we did not want our students to become computer experts while sacrificing their ability to solve problems with paper and pencil. It is obvious, however, that one cannot introduce a computer supplement into a finite mathematics course without making some adjustments. Since we were restricted to the same 45 contact hours as in the standard course, we had to eliminate certain introductory topics mentioned at the beginning of this paper; however, we did not reduce the amount of time devoted to probability and matrices. We eliminated enough material to give us an additional eight contact hours, and we found this additional time sufficient for the discussion of computing in general and the computer assignments in particular. If, on the other hand, one wished to add a computer supplement without sacrificing any of the standard course material, the best approach would probably be to increase the contact hours per week from 3 to 4. The extra hour per week would be more than adequate to accommodate the computer supplement.

We also believe it is important for the students and the teacher to maintain a sense of perspective about the goals of this kind of course. It is too easy to drift unintentionally into a computer programming seminar to the detriment of the traditional theoretical and problem-solving aspects of a finite mathematics course. The student must be kept mindful of the main thrust of the course, even if, as is often the case, he is more interested in computers than in mathematics. Keeping the course properly balanced becomes a question of the instructor's resolve, but he can help himself (as we did) by keeping the computer part of the course out of the hour tests and final examination.

Finally, we believe that when computer assignments are returned to the student, the student's program should be accompanied by the instructor's solution. The solution can be typed in capitals to resemble the actual print-out obtained at a teletypewriter terminal. Besides showing the student the correct answer, an instructor's solution provides the student with a valuable reference if he pursues computer programming after completion of the present course. Such a policy, of course, requires new or altered computer assignments for subsequent courses, but we consider the price small in relation to the benefits.

We must admit, in summary, our satisfaction with the results of this course. It will likely be a permanent part of our curriculum, and we anticipate further refinements and additions to the computer assignments in future offerings of the course. To this end, we invite others who have had similar experiences with a computer-supplemented finite mathematics course to share their views with us.

REFERENCES

1. Campbell, H.G. and Spencer, R.E., Finite Mathematics, New York: Macmillan, 1974.
2. Kemeny, J.G., et. al., Finite Mathematical Structures, Inglewood Cliffs: Prentice Hall, 1959.
3. Kemeny, J.G. and Snell, J.L., Finite Markov Chains, Princeton: Van Nostrand, 1960.
4. Mendenhall, W., Introduction to Probability and Statistics, 3rd edition, Belmont, California: Wadsworth, 1971.
5. Mosteller, F., Fifty Challenging Problems in Probability, Reading, Pennsylvania: Addison-Wesley, 1965.

A COMPUTER-ASSISTED COURSE IN BIOMATHEMATICS

Pui-Kei Wong
Mathematics Department
Michigan State University
East Lansing, Michigan 48824
517-353-6880

Applications of mathematics to problems in the physical sciences and engineering are well known and have long been an integral part of the undergraduate curriculum of mathematics departments. During the past two decades courses in numerical methods and elementary finite mathematics have also been added by many colleges and universities, reflecting the growing importance of mathematics in economics, management, and social sciences as well as in modern technology. We need only cite the contributions of such individuals as Kenneth Arrow, George Dantzig, Wassily Leontiff, Paul Samuelson, and John von Neumann in this connection.

On the other hand, applications of mathematics to the biological and life sciences, though no less important, have not found their proper place in the undergraduate curriculum. It was J.B.S. Haldane who observed in 1928 that: "The permeation of biology by mathematics is only beginning, it will continue and (grow) into a new branch of applied mathematics." In the decades since, we have witnessed a phenomenal growth in the applications of mathematics to the agricultural, biological, and life sciences, beginning with the work of Haldane, Lotka, Volterra, and Wright and onto that of Hodgkin and Huxley, who shared the Nobel prize in physiology or medicine in 1963. And most recently, Allan Cormack and Godfrey Hounsfield were awarded the 1979 Nobel prize in physiology or medicine for their pioneering work in applying mathematics and computer technology in the x-ray image reconstruction technique called Computerized Axial Tomography (1).

Recognizing the growing impact of mathematics in the life sciences, the Mathematics Department at Michigan State University introduced five years ago a new course MTH-481 titled "Selected Mathematical Ideas in Biology." This is a one-quarter, four-credit course with calculus as prerequisite. At Michigan State we have a two-term, ten quarter-credit sequence in calculus for the biological and social sciences. A student having completed this abbreviated calculus sequence is usually adequately prepared to enter MTH-481. No prior knowledge of computer programming is assumed.

The major objective of this course is to provide the student with a sound introduction to mathematical methods and deterministic models in biology, especially in the use of computer simulations to analyze model behavior. To this end a number of problems from biology are introduced and studied in some depth. We start with the underlying biological description, hypothesize, and then derive a working mathematical model. Once constructed, the model can be manipulated and analyzed to reveal its possible behavior. Modifications on the basic model are made, and the system is then studied again under various initial conditions, external inputs, and perturbations. All this can be done quickly and economically by computer simulations. Instead of numerical solutions, it is often more desirable that the long-term trend or qualitative behavior and stability properties of the model be known. In this case computer graphics provide a particularly effective tool in the analysis of model dynamics.

In addition to conventional classroom lectures and homework, students are required to spend time each week in the computer laboratory doing modeling and simulation exercises. These exercises use instructional modules written specifically for the course, and they run on Tektronix 4051 graphics computing systems. These stand-alone microcomputers are capable of high resolution graphics (780 lines by 1024 pixels per line) and are especially well suited to our needs. A Tektronix

4631 hardcopy unit provides a record of the graphic output of any desired simulation run.

We also use computer graphics in the classroom. Demonstrations are done using a Tektronix 4025 graphics terminal and an Advent 1000A television projection system. The 4025 is a raster scan device with 480 x 640 resolution and provides a composite video output to the Advent. It is driven by a Tektronix 4051 computer. The Advent has a seven-foot diagonal screen and is suitable for viewing by a class of up to forty students. Dynamics of a model can be displayed on the screen before a class and discussed very effectively this way. A Tektronix 4662 digital plotter is also used to make high quality multicolored transparencies for use on ordinary overhead projectors.

The mathematics treated in MTH-481 include matrix algebra, difference, and differential equations, and applications are drawn from various areas of biology. Special emphasis is given to the qualitative behavior and stability of nonlinear equations. The course will typically cover the first twenty-two items of the outline below plus occasional substitution from the remaining ones.

MTH 481 Syllabus

1. Algebra of vectors and matrices.
2. Linear equations and determinants.
3. Eigenvectors and eigenvalues.
4. Methods of least squares.
5. Discrete time single species population models.
6. Linear difference equations.
7. Depletion of nonrenewable resources.
8. Discrete time predator-prey models.
9. Populations with age structure (Leslie model).
10. Harvesting and exploitation of renewable resources.
11. Population genetics and difference equations.
12. Asymptotic behavior and stability (period three implies chaos).
13. Analysis of growth data.
14. Logistic, Gompertz, and other continuous time population models.
15. Geometrical analysis of differential equations.
16. Models of photosynthesis.
17. Linear differential equations and systems.
18. Compartmental analysis.
19. Lotka-Volterra and other models of competition and interaction.
20. Volterra-Gause principle.
21. Theory of chemostat.
22. Stability and limit cycles.
23. Enzyme kinetics.
24. Simple epidemic models.
25. Cell cycle analysis.
26. Models in neurophysiology.
27. Fishery dynamics and marine food chains.

The interactive computer-assisted instructional modules used in the course are designed for the student with no prior programming experience, and they are stored on data tapes. An orientation lecture on the use of the Tektronix graphics computing system is given at the start of each term, explaining how the student can access the system and use these instructional tapes. These modules range from tutorials on vectors and matrices through population growth, fitting data to logistic growth curve, differential equations, and modeling using compartmental analysis and the method of peeling. Students have found using interactive computer graphics particularly helpful and have later adapted and modified some of the programs written for this course for their own research use.

To illustrate the use of computer simulation in teaching mathematical modeling, let us consider the problem of growth of human populations. First the problem is analyzed by the very simple difference equation

$$x_{k+1} = Ax_k \, , \; k = 0,1,2,\ldots \tag{1}$$

Here x_k is the population size or density at the kth census and A is the intrinsic growth rate, which is defined as the difference between the crude birth rate B and crude death rate D. In this case the population is treated as a homogeneous collection of individuals so age structure and sexual differences are ignored. Equation 1 admits the closed form solution

$$x_k = A^k x_0 \, , \; k = 0,1,2,\ldots \, , \tag{2}$$

where x_0 is the initial population if A is constant. The future behavior of the population is therefore completely determined by the magnitude of A. However real populations can rarely be described realistically by such a simple model in which the growth rate is constant over long periods of time.

The first modification we make then is to allow the intrinsic growth rate A to be time- or population-dependent. Closed form solutions to Equation 1 are in general no longer possible, but the problem is easily handled by the computer. In the CAI module for this course, actual census data is used to construct population projections. Figure 1 shows several different functions used to fit the crude birth rate data for Costa Rica. The student is asked to select one of these, and the computer will then calculate and display the corresponding population trend for a specified number of years into the future (see Figure 2). At the end of each simulation run the student has the choice of changing the parameters or entering an entirely different birth rate function and running the problem again. In this case the same death rate function is used for all the simulation runs.

At the next level of complexity, a population is studied by separating it into sex and age classes using the basic model of Leslie (2). For human populations where the ratio of males to females is essentially constant over long time periods, it is customary to consider only the females and divide them into five-year age classes and take census every five years. The basic equation is still Equation 1, but x_k is now a population vector with n components, and A is the projection matrix.

$$A = \begin{bmatrix} f_1 & f_2 & f_3 & \cdot & \cdot & \cdot & f_n \\ s_1 & 0 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & s_2 & 0 & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & \cdot & & & & \cdot \\ 0 & 0 & 0 & \cdot & \cdot & \cdot & s_n \end{bmatrix} .$$

Here f_j is the average number of female offspring born to a member of the jth class every five years, and s_j is the probability that a member of the jth class will survive and advance the $(j+1)$st class at the end of five years. If we restrict our attention to females of ages 0 - 59, there will be twelve age classes and A will be a 12×12 matrix. If we assume constant fecundities and survival probabilities, the population for subsequent years can be calculated using Equation 2 as in the simple scalar case. Figures 3 - 6 are the computer-generated population profiles for American females based on the 1964 data for fecundities $f_1, \ldots, f_{12}$ and survival probabilities $s_1, \ldots, s_{12}$.

In the final stage the effects of population-dependent A as well as harvesting are added to the Leslie model and studied. Figure 7 shows the menu selection for this model, and Figures 8 and 9 are the output of two different simulation runs for a hypothetical population having three age classes. Extensions of the model to include insects, trees, and other populations that are more naturally grouped into developmental stages or size classes are also included at this stage.

All the CAI material was written in TEK-BASIC specifically for interactive use and has been revised and tested over the past five years. Although much of it was designed with teaching mathematical modeling to biologists in mind, some of it can be and has been adapted and modified for use in other courses as well. The equipment was acquired in part with funds from the National Science Foundation under an Instructional Scientific Equipment Grant.

REFERENCES

[1] L.A. Shepp and J.B. Kruskal, "Computerized Tomography: The New Medical X-ray Technology," American Mathematical Monthly, 85 (1978), 420-439.

[2] P.H. Leslie, "On the Use of Matrices in Certain Population Mathematics," Biometrika, 33 (1945), 183-212.

Fig. 1

```
CRUDE BIRTH RATE FOR COSTA RICA,1930-2050
VERTICAL SCALE = NO. OF BIRTHS PER 1000 POPULATION
DIAMONDS REPRESENT CENSUS DATA
LINE L IS LEAST SQUARE LINEAR FIT TO DATA
P1, P2 & P3 ARE THREE MODIFIED EXPONENTIAL FITS
WHICH PROJECTION CURVE DO YOU WANT TO USE? <L, P1, P2 OR P3>
```

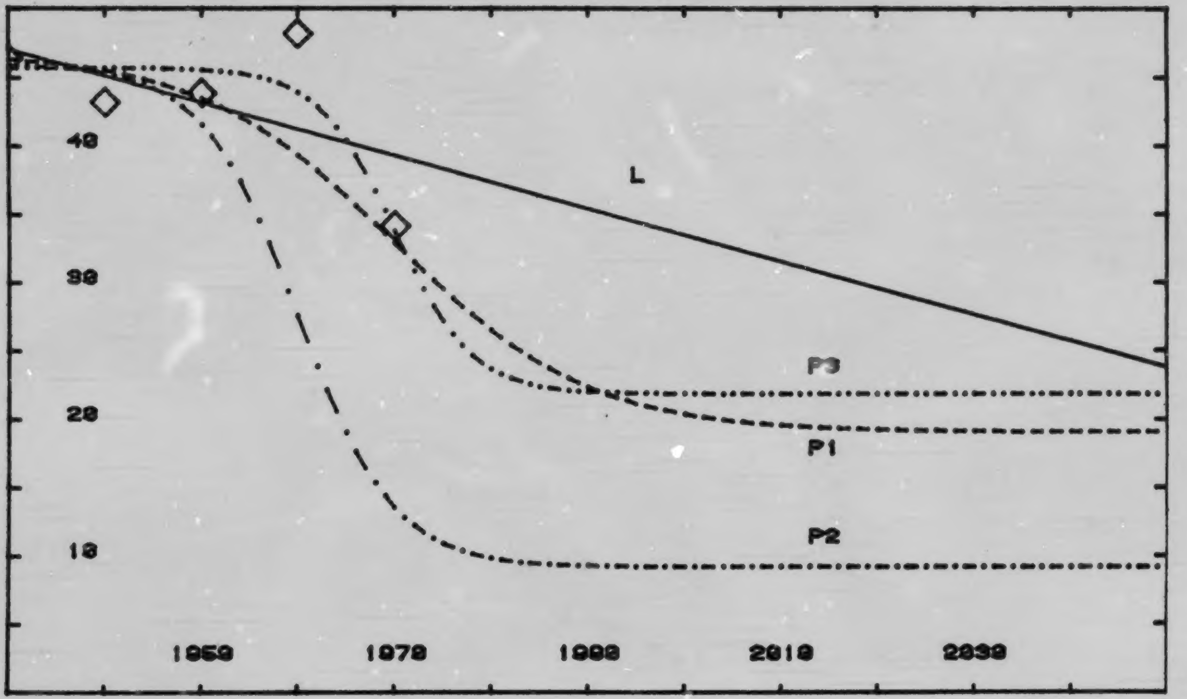
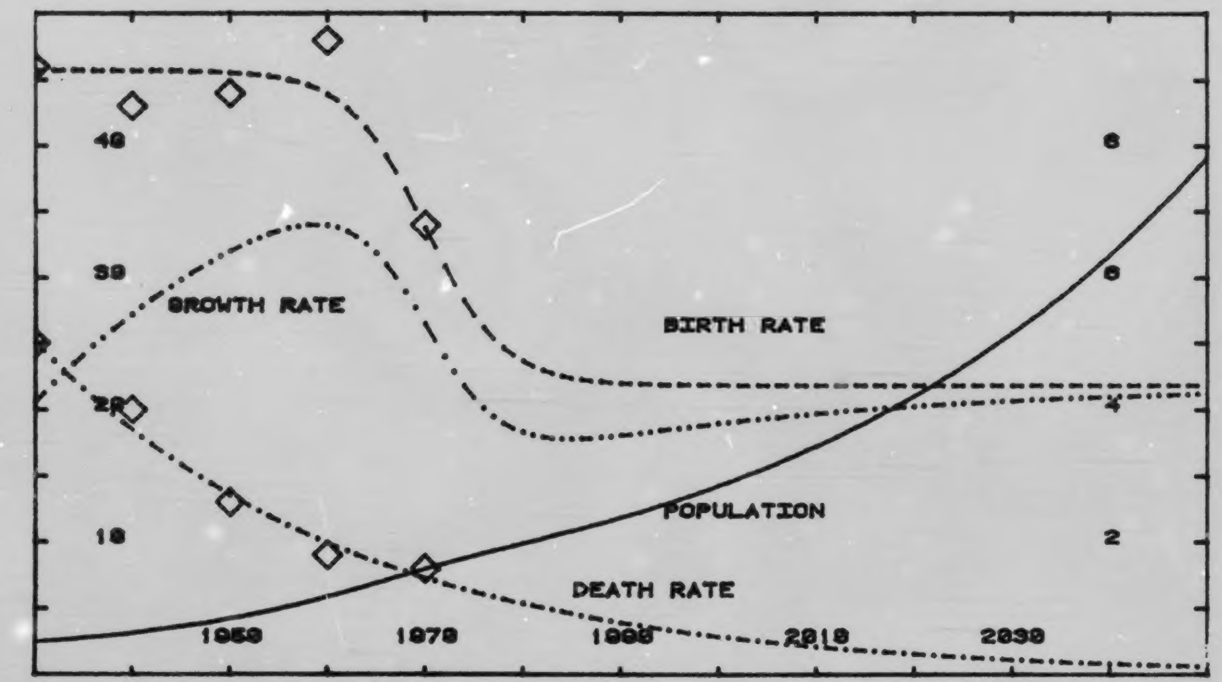


Fig. 2

POPULATION OF COSTA RICA, 1930-2050, BASED ON P1 PROJECTION

RIGHT VERTICAL SCALE = POPULATION IN MILLIONS (SOLID CURVE)
LEFT VERTICAL SCALE = RATES PER 1000 POPULATION
DIAMONDS REPRESENT ACTUAL CENSUS DATA



AMERICAN FEMALES, AGES 0-59
POPULATION PROFILE AT T = 1964

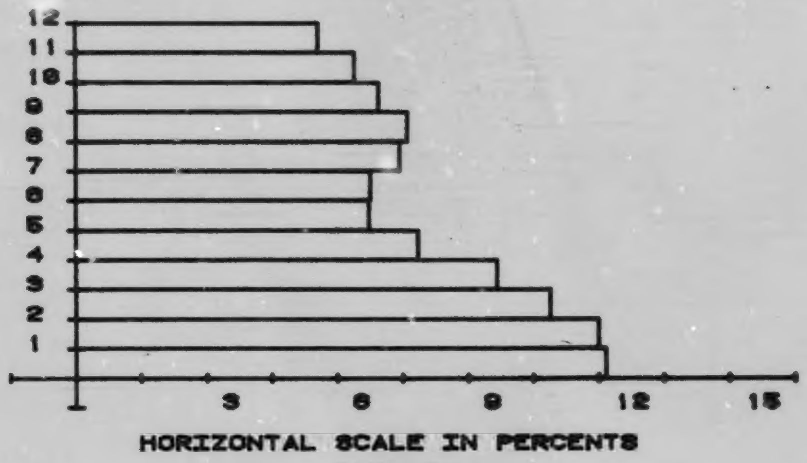


FIG. 3

AMERICAN FEMALES, AGES 0-59
POPULATION PROFILE AT T = 1969

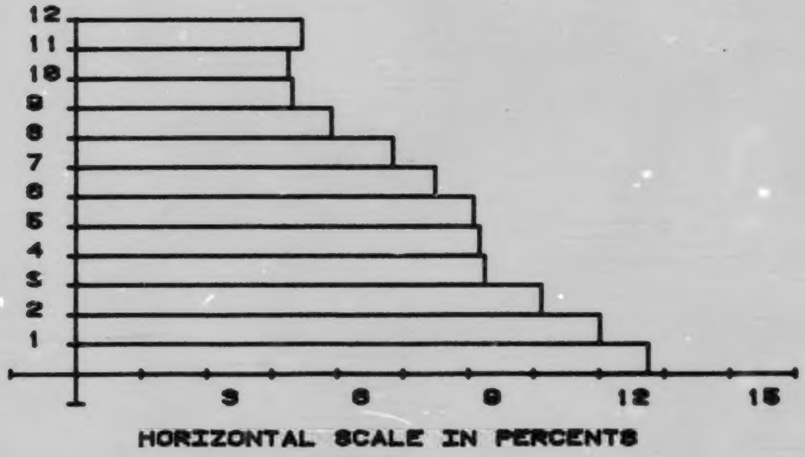


FIG. 4

AMERICAN FEMALES, AGES 0-59
POPULATION PROFILE AT T = 2014

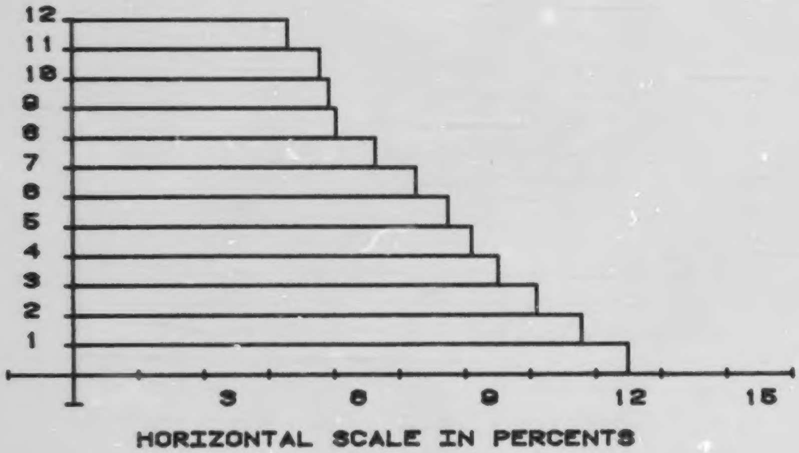


FIG. 5

AMERICAN FEMALES, AGES 0-59
POPULATION PROFILE AT T = 2039

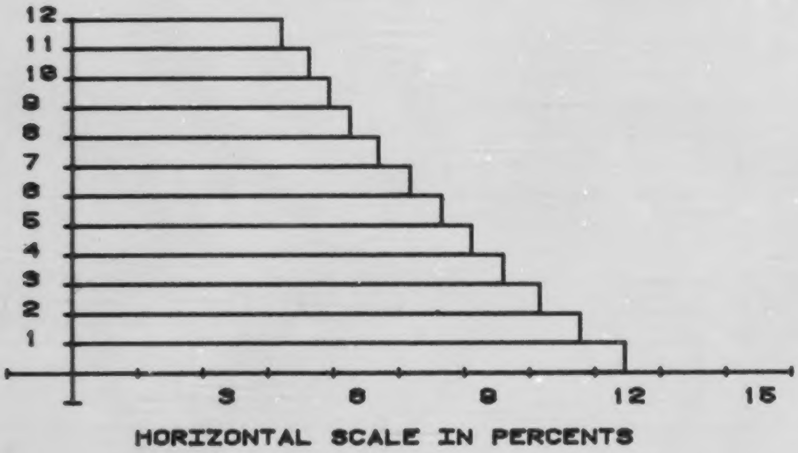


FIG. 6

Fig. 7

```
     *** MENU ***

1.  CONSTANT FECUNDITY AND SURVIVAL PROBABILITY

2.  POPULATION DEPENDENT FECUNDITY, CONSTANT
    SURVIVAL PROBABILITY

3.  CONSTANT FECUNDITY, POPULATION DEPENDENT
    SURVIVAL PROBABILITY

4.  POPULATION DEPENDENT FECUNDITY AND SURVIVAL
    PROBABILITY

5.  HARVESTING


SELECT BY NUMBER THE ITEM YOU WANT:   3
```

```
LESLIE'S MODEL: CONST. FEC., POP. DEPENDENT SURV. PROB.
FND(W) = 1/(1+EXP(W/D1-D2)),  D1 = 200  D2 = 2.5
F1 = 0              F2 = 12             S1 = 0.333333333  S2 = 0.5
X(0) = 6            Y(0) = 4            Z(0) = 1              W(0) = 11
PLOT OF TOTAL POPULATION W(K)
```

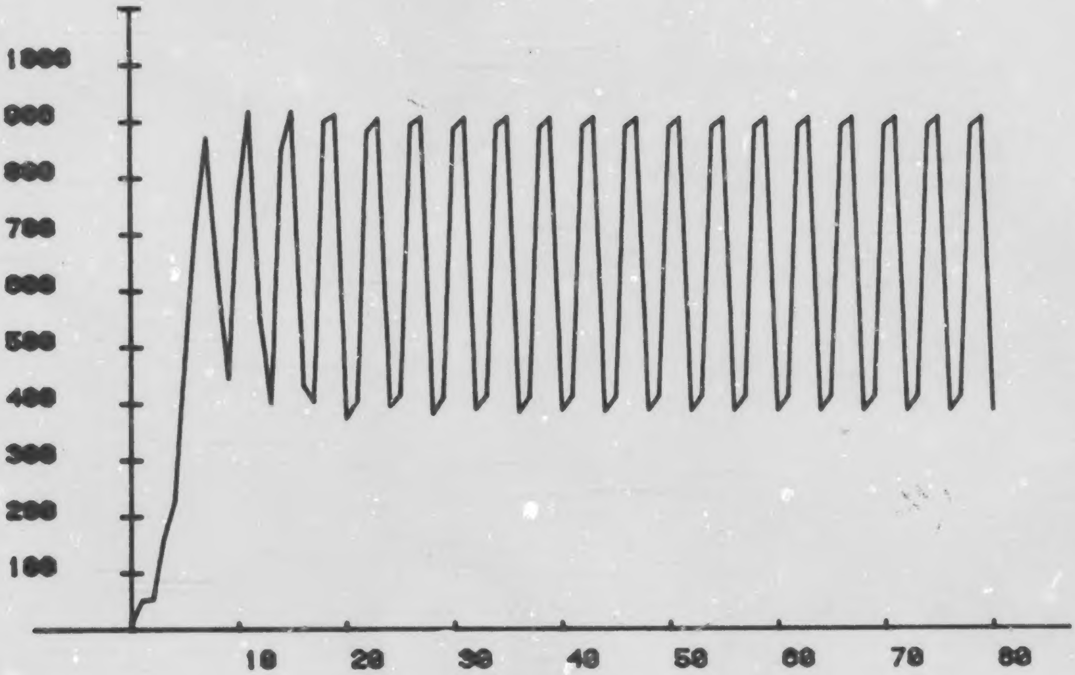


FIG. 8

```
LESLIE'S MODEL: CONST. FEC., POP. DEPENDENT SURV. PROB.
FND(W) = 1/(1+EXP(W/D1-D2)),  D1 = 100  D2 = 5
F1 = 0              F2 = 12               S1 = 0.9999999999 S2 = 0.5
X(0) = 6            Y(0) = 4              Z(0) = 1             W(0) = 11
PLOT OF TOTAL POPULATION W(K)
```

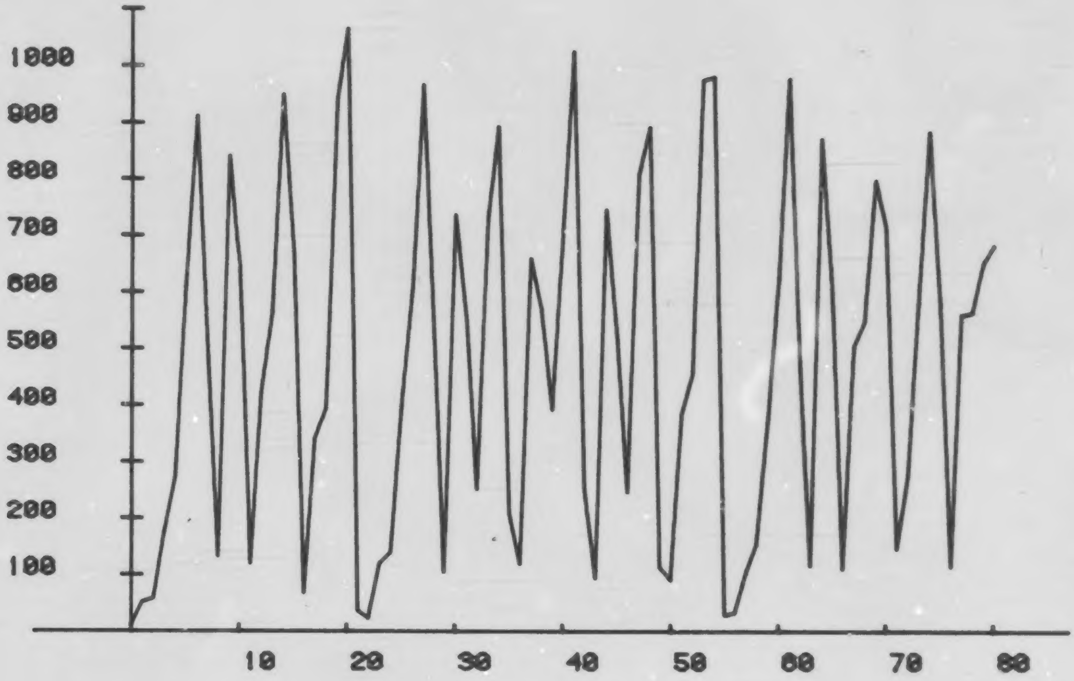


FIG. 9

COMPUTER SYMBOLIC MATH

David R. Stoutemyer
Electrical Engineering Department
University of Hawaii at Manoa
Honolulu, Hawaii 96822
(808) 948-8196

Most current calculators and mathematical computer programs are oriented toward approximate numerical computations using an occult arithmetic called chopped nondecimal fixed-precision floating-point. If this revelation causes discomfort, consider what Anston Householder, dean of numerical analysts, said: "I don't like to fly in airplanes, knowing they are designed with floating-point arithmetic."

This arithmetic has undeniable advantages, but it also has decidedly bizarre implications which are difficult to convey even to graduate numerical analysis students. More important, it is not the arithmetic that any of us uses for manual computation.

THE NATURE OF COMPUTER SYMBOLIC MATH

The good news is that computers also can do exact rational arithmetic. Moreover, even small personal microcomputers can do the nonnumeric symbolic operations of algebra, trigonometry, analytic geometry, and calculus. In fact, use of such computers for these purposes does not even require a knowledge of computer programming, other than the widespread custom of using "*" to denote multiplication and "↑" to denote raising to a power, in order to unambiguously specify expressions using a one-dimensional format compatible with the limitations of standard keyboards and displays. For example, here is a sample interactive dialogue using the muMATH-79™ symbolic math program on an inexpensive Radio Shack TRS-80™ computer, which is increasingly popular in schools:

In order to expand the expression $(5x-7)(2y+3)^4$ the user types the line

```
(5*X-7)*(2*Y+3)↑4;
```

and a few seconds later the interactive response is

```
80*X*Y↑4 - 112*Y↑4 + 480*X*Y↑3
 -672*Y↑3 + 1080*X*Y↑2 - 1512*Y↑2
 +1080*X*Y - 1512*Y + 405*X - 567
```

Next, to solve the equation $2(x^3-a^2x) = x^3 - a^2x$ for x, the user types

```
SOLVE(2*(X↑3-A↑2*X) = X↑3-A↑2*X, X);
```

and the interactive response is the solution set

```
{X=0,
 X=A,
 X=-A}.
```

Next, to invert the matrix

$$\begin{Bmatrix} 1 & p \\ 0 & q \end{Bmatrix},$$

the user types

```
{[1,P]
 [0,Q]} ↑ -1;
```

and the response is

```
{[1, P/Q]
 [0, 1/Q]}.
```

Then, to simplify the trigonometric-logarithmic expression $(\tan a)\cos a)+ 1/\csc(a)+\ln(x^2y) - 2\ln x$, the user types

```
TAN(A)*COS(A) + 1/CSC(A) + LN(X↑2*Y)
  -2*LN(X);
```

yielding the response

```
2*SIN(A) + LN(Y).
```

To sum the series

$$\sum_{j=1}^{n} (a\ j^2 + b^j)$$

the user types

```
SUM(A*J↑2+B↑J, J, 1, N)
```

to get the response

```
(2*A*N↑3+3*A*N↑2+A*N)/6
  + (B↑(N+1)-B)/(B-1).
```

(The arguments of the sum function are respectively the summand, summation index, lower limit, and upper limit.) Finally, to evaluate the integral

$$\int [ax^2 + x \sin x^2] dx ,$$

the user types

```
INT(A*X↑2 + X*COS(X↑2), X)
```

yielding

```
A*X↑3/3 + SIN(X↑2)/2.
```

In contrast, traditional programming languages such as APL, BASIC, FORTRAN or PASCAL provide built-in math facilities essentially only for limited-precision *arithmetic*.

The above examples have illustrated symbolic math capabilities relevant to grades 8 through 14. However, computer symbolic math also features exact rational arithmetic, which is more suitable than floating-point for supporting math education at the kindergarten through sixth grades. For example, continuing the above dialogue, to simplify the expression $30^{30}\sqrt{12}/40!$ the user types

```
30↑30 * 12↑(1/2)/40! ;
```

and the response a few seconds later is

```
(253410816192626953125/
  502114286731006316278912)*(3)↑(1/2)
```

There are none of the *roundoff*, *underflow*, or *overflow* problems that beset traditional mathematical programming languages.

Although none of the above examples requires a knowledge of computer programming other than the convention of using "*" for multiplication and "↑" for raising to a power, curiosity of needs not met by the built-in facilities causes some users to seek deeper involvement with the mathematical and programming techniques used to implement these symbolic math systems. Accordingly, symbolic math systems generally provide a programming language for writing extensions. Examples of such extensions are:

1. introducing new functions (such as hyperbolic functions) and their simplification properties;
2. extending the class of expressions which can be factored, differentiated, or otherwise operated upon;
3. extending the class of equations which can be solved;
4. automating the sequence of steps necessary to do an inductive proof, determine a limit, test a series for convergence, or determine the first few terms of a Taylor series.

EDUCATIONAL USES

Computer symbolic math has long been available to applied mathematicians who have generous computing allowances on the largest computers, but this tool has only recently become available on the inexpensive small computers most often available to kindergarten through sophomore calculus classes. How can this increasingly available tool support math education?

1. A symbolic math system can be used by a computer-aided instruction program to provide far more flexible, intelligent, and responsive automatic drill or examination than is otherwise possible. For example, symbolic math systems are able to recognize the equivalence of a wide variety of mathematically equivalent expressions, and a computer-aided math instruction program can adaptively take alternate courses of action depending upon the users' performance. Such programs free teachers to do what they do best--provide warmth, understanding, individual high-level guidance, and assistance for unanticipated difficulties.
2. The ability to do numerous large examples encourages creative explorations which can reveal patterns and thus suggest general theorems. Students' inductive generalization powers can be exercised and developed to a far greater degree than is otherwise possible.
3. Built-in *trace* facilities can allow students to see each step of a computation, rather than merely the final result.
4. A demonstration that an operation can be done automatically by computer can encourage average and poor students that the flashes of inspiration given only to brilliant students are unnecessary for that operation. Hope for plodders is revealed.
5. Students who are more enthusiastic about computers than about math are provided with a new avenue for appreciation of math. Computer symbolic math vastly enhances the opportunity for beneficial mutual reinforcement and cross-motivation between math and computers.

6. Inspection of the underlying computer symbolic math implementation programs can help students learn the methods for accomplishing the operations manually.
7. Programming extensions to the built-in operations can reinforce understanding of both the built-in and new operations. In fact, an instructor can withhold portions of the algebra system implementation and challenge the students to implement them.

AVAILABILITY

For educational use, a symbolic math system should be interactive, general purpose, and available for a modest fee on computers typically available to students. In order of increasing computer memory requirements, here are four symbolic math packages which meet these requirements:

PICOMATH-80^{tm} is a set of three small symbolic math demonstration programs written in BASIC that should run on virtually any computer. For information, write The Soft Warehouse at Box 11174, Honolulu, Hawaii 96828.

muMATH-79 is currently available for many of the various brands of personal microcomputers based on the Intel 8080, Intel 8085, and Zilog Z80 microprocessor chips. Computer memory can be measured in units called *bytes*, and muMATH requires from 32,000 to 64,000 bytes of memory, depending on how many of the various mathematical facilities are loaded simultaneously. For information, write Microsoft, at 10800 N.E. 8th, Suite 819, Bellvue, Washington 98004.

FORMAC is available for medium to large IBM 360 and 370 computers which have at least 140,000 bytes of memory. For information, write Knut Bahr at GMD/IFV, D-6100, Darmstadt, Germany.

REDUCE is currently available for IBM 360 and 370, DEC PDP 10 and 20, Univac 1100 series, CDC Cyber series, and Burroughs 6700 computers. REDUCE requires a minimum of from 300,000 to 400,000 bytes depending on how many of the various mathematical facilities are loaded simultaneously. For information, write Professor Anthony C. Hearn at the Computer Science Department, University of Utah, Salt Lake City, Utah 84112.

LITERATURE

Most of the relevant literature is oriented toward research rather than education, but a good way to get started in this area is to join the Association for Computing Machinery Special Interest Group on Symbolic and Algebraic Manipulation. Their *ACM SICSAM Bulletin* is a timely source of information on such topics as meetings, abstracts, and new systems. For information about joining, write the ACM at 1133 Avenue of the Americas, New York, NY 10036.

Invited Sessions

COMPUTER-BASED RESOURCE SHARING:
DIVERSITY AND OPPORTUNITY

Chaired by Donna Davis Mebane and Rodney Mebane
EDUCOM
P.O. Box 364
Princeton, NJ 08540
(609) 921-7575

ABSTRACT

Computer-based resource sharing occurs in many places, by many people, in many forms, for many reasons. A professor who asks a colleague to evaluate the pedagogic quality of a CAI tutorial, a student user of SPSS, a demographer working with census tapes, a librarian with access to OCLC -- all of these people are engaged in some form of computer-based resource sharing. This special session will report on a major EDUCOM research project, funded by NSF, to examine the diversity of sharing activity that takes place within the higher education and research community and the opportunity that exists for increased cooperation.

In describing, explaining, and evaluating the sharing phenomenon, two scenarios are apparent. One is of relatively small-scale activity, very informal and with information exchanged primarily by word of mouth. Many are neither aware of what others are doing with computers nor of the possibilities for shared use. The other scenario is of an active attempt by various resource-sharing organizations (RSOs) to promote the orderly exchange of computing materials and experiences.

Focusing on the latter scenario, this session will begin with the benefits of sharing and the various pathways to sharing. It will then specifically address interinstitutional resource exchange and nine representative organizations that provide a link between resource developers and a resource users and effect the shared use of hardware, software, machine-readable data bases, and other computing systems. The RSOs participating in the EDUCOM study include the following nonprofit organizations:

- CONDUIT
- EDUNET
- Georgia Information Dissemination Center (GIDC)
- Health Information Network (HEN)
- Inter-university Consortium for Political and Social Research (ICPSR)
- Merit
- New England Regional Computing Program (NERComP)
- North Carolina Educational Computing Service (NCECS)
- Research Libraries Information Network (RLIN)

Profiles of these organizations will be presented, and the nature of services offered will be explored in depth. Specific RSO activities will be cited to illustrate the diversity of organizational response to such questions as these:

- What kinds of resources are available and where do they originate?
- What options do developers have?
- How are resources packaged and distributed?
- Who are the current users of these organizations and what services do they find most valuable?
- What are the technical, economic, and organizational conditions of using shared resources?
- How do the sharing organizations themselves adapt to a rapidly changing environment?

The primary objective is to make practitioners and leaders in the field more aware of organizational options available to meet their own diverse computing needs.

Materials will be distributed describing the study and the participating organizations, and representatives of the nine organizations have been invited to apply for space in the exhibit area.

COMPUTERS AND INSTRUCTION:
DEVELOPMENT, DIRECTIONS, AND ALTERNATIVES

Chaired by William Gruener
Addison-Wesley Publishing Co., Inc.
Reading, MA 01867
(617) 944-3700

ABSTRACT

This session will examine the world of computers in the instructional environment. Our three panelists have used micros, minis, and mainframes in various educational situations and will present papers as a basis for discussion and examination of computers and instruction.

During the session directions and alternatives for new uses of computers will be explored and the existing systems examined. Participants and attendees are encouraged to exchange information and ideas. It is hoped everyone involved will leave with fresh insight into the creative use of computers for educational purposes.

CREATIVE LEARNING WITH COMPUTERS--THEORIZE OR PERISH

Margot Critchfield
Pittsburgh, PA

ABSTRACT

The experience of the dyed-in-the-wool computer hobbyist provides educators one point of reference for our warm enthusiasm for the computer. Our observation of students who learn to program or who play complex games on interactive microcomputers provides another point of reference. What do these vastly different kinds of learners have in common? A theory of learning is needed to tie these disparate computer users together. Such a theory must be applied to the design of educational experiences that include computers in order to ensure computers' success and continued growth.

This paper will attempt to clarify the definition of creative learning and to state some of the explicit values of formal education as they relate to computer use. The possible contribution of some current learning theories to a set of principles for creative learning with computers will be discussed.

PLATO: COMPUTERS AND INSTRUCTION, A LARGE-SCALE SYSTEM

Robert Hart
University of Illinois

ABSTRACT

The Language Learning Laboratory at the University of Illinois makes available an 80 terminal PLATO site for humanities usage. During the past seven years, the Language Learning Laboratory has supported PLATO materials development for French, German, Spanish, Russian, Swedish, Swahili, Hindi, Hebrew, Chinese, Japanese, Latin, Classical Civilizations, and E.S.L. The relative independence of development projects has led to a number of models for incorporating PLATO in classroom activity, ranging from totally PLATO-centered to voluntary and supplementary usage. Research now in progress seems to reveal a relatively stable pattern across a wide variety of situations:

(1) There are two overlapping but distinct populations of users, one PLATO-receptive, the other non-receptive.

(2) Both instructor and student dissatisfaction with current materials centers on response analysis and feedback characteristics, which are perceived to be relatively inaccurate, inflexible, and unable to deal with meaning as opposed to form.

MICROCOMPUTERS IN ELEMENTARY AND SECONDARY EDUCATION: WHERE WE'VE BEEN, WHERE WE'RE GOING.

Dan Isaacson
University of Oregon

ABSTRACT

What educational research says about the use of computers in teaching, what's happening now, what the future holds, and what's holding us back will be discussed.

Tutorial

VIDEODISC TUTORIAL

Bobby R. Brown and Joan Sustik
Weeg Computing Center
University of Iowa
Iowa City, Iowa 52242
(319) 353-[illegible]170

ABSTRACT

This tutorial is for individuals who have little or no experience using a videodisc in instruction. The tutorial will consist of a presentation of the performance characteristics of videodisc as they relate to instruction. A variety of applications current and potential will be discussed. Various approaches to the development of materials and mastering of discs for videodisc applications will be presented. The tutorial will also include a demonstration of an operational intelligent videodisc system. Handouts will be provided to the participants.

Testing/Placement

MICROCOMPUTER-ASSISTED STUDY AND TESTING SYSTEM (MASTS)

Hugh Garraway
The University of Texas at Austin
EDB 438B
Austin, Texas 78712
(512) 471-4014

INTRODUCTION

Computer-managed instruction (CMI) has proven to be an effective teaching/learning strategy. Studies using large computer-based instruction (CBI) systems such as PLATO (Nievergelt, Jurg and others, 1978) and TICCIT (Reigeluth, 1978) have shown that in addition to facilitating learning, students enjoy CMI. The military is one of the largest implementers of CMI. The Navy is using several TICCIT systems in a CMI application that is effective in terms of both cost and instruction. The Air Force is using a dedicated CMI system, the Advanced Instructional System (AIS), developed with the McDonnell Douglas Corporation.

The Teaching Information Processing System (TIPS) and the Program for Learning in Accordance with Needs (PLAN) are two CMI systems available for use in all levels of education. Both of these systems are batch-oriented and are intended to be used on mainframe computers. TIPS and PLAN are designed to be used mainly with large numbers of students.

Successful medium- and small-scale applications of CMI (Bork, 1977; Brockmeier, 1977) have helped define some common problems in using CMI including:

1) Time and expense involved in custom designing CMI systems for individual applications.
2) Difficulty with adapting existing CMI programs to run on computer systems other than those for which they were developed.
3) Knowledge of programming necessary for an instructor to create or modify a program.
4) High initial cost for setting up hardware or adding to existing systems.
5) Communication expenses and problems, and down time associated with time-sharing systems.

Recommendations for improving the cost effectiveness and efficiency of future CMI systems have included the use of stand-alone microcomputer systems and advanced authoring languages that allow an instructor to specify or choose instructional and testing strategies and enter related content without having to learn a complicated programming language.

PROJECT OBJECTIVE

The objective of this project was to design, produce, implement, and evaluate a CMI system with the following specifications:

1) The system will be based on inex-

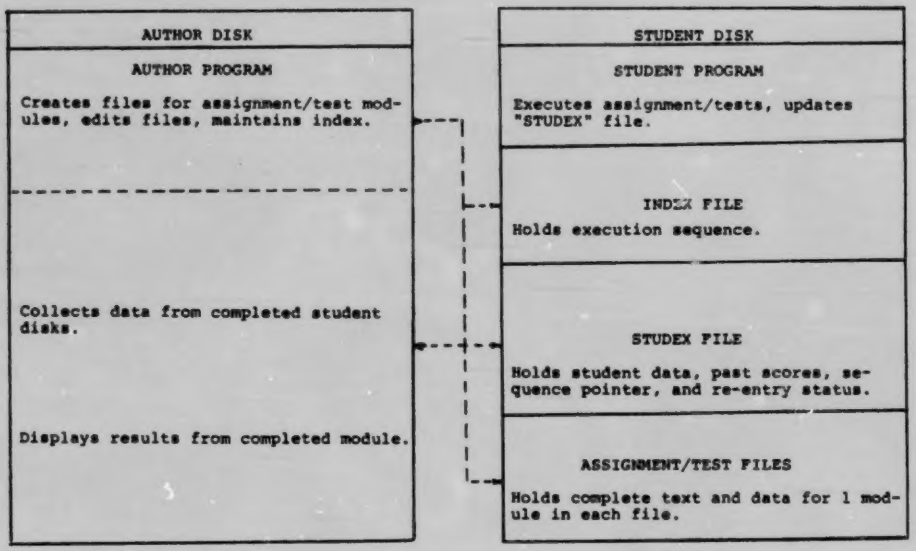

AUTHOR DISK
AUTHOR PROGRAM
Creates files for assignment/test modules, edits files, maintains index.
Collects data from completed student disks.
Displays results from completed module.
STUDENT DISK
STUDENT PROGRAM
Executes assignment/tests, updates "STUDEX" file.
INDEX FILE
Holds execution sequence.
STUDEX FILE
Holds student data, past scores, sequence pointer, and re-entry status.
ASSIGNMENT/TEST FILES
Holds complete text and data for 1 module in each file.

FIGURE 1

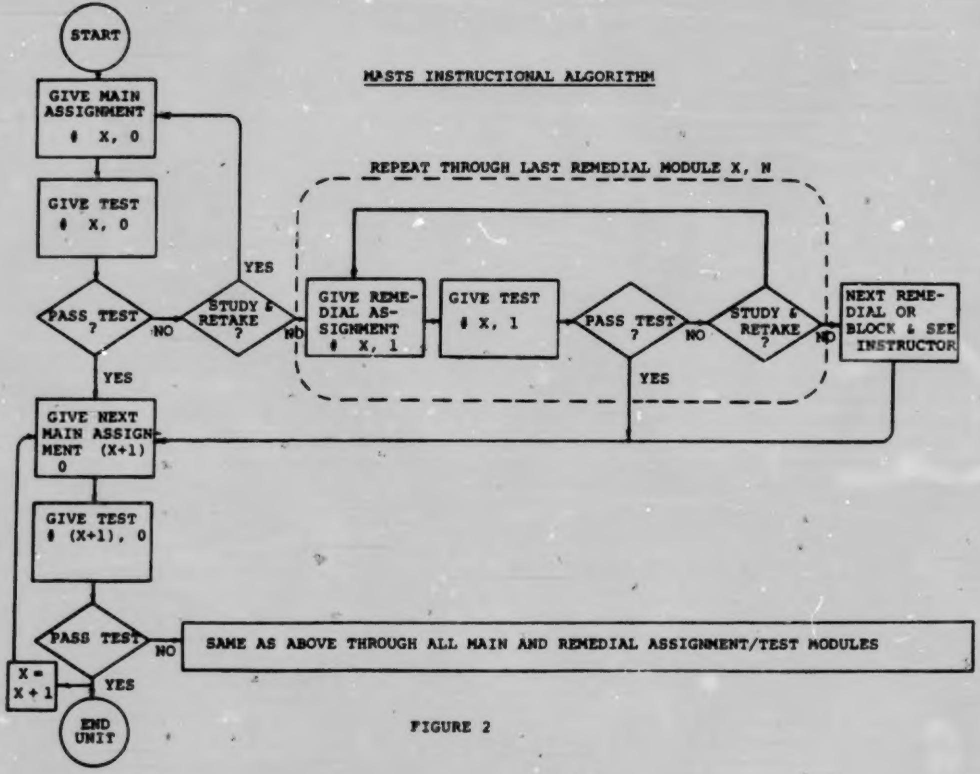

START
MASTS INSTRUCTIONAL ALGORITHM
GIVE MAIN ASSIGNMENT # X, 0
GIVE TEST # X, 0
PASS TEST ?
NO
YES
STUDY & RETAKE ?
YES
NO
REPEAT THROUGH LAST REMEDIAL MODULE X, N
GIVE REMEDIAL ASSIGNMENT # X, 1
GIVE TEST # X, 1
PASS TEST ?
NO
YES
STUDY & RETAKE ?
NO
NEXT REMEDIAL OR BLOCK & SEE INSTRUCTOR
GIVE NEXT MAIN ASSIGNMENT (X+1) 0
GIVE TEST # (X+1), 0
PASS TEST
NO
YES
SAME AS ABOVE THROUGH ALL MAIN AND REMEDIAL ASSIGNMENT/TEST MODULES
X = X + 1
END UNIT

FIGURE 2

pensive microcomputers with disk storage ability.

2) The system will be interactive.

3) The CMI program(s) will be contained on individual student disks so that they may be used on any compatible microcomputer.

4) An interactive, intelligent authoring program will allow an author to create CMI modules without having to master a programming language.

5) The system will allow an author to gather student data from individual disks for storage on a single data disk.

6) The system will perform simple statistical computations and display group and individual results as raw score, mean, standard deviation, and z-score.

7) The collected data will be stored in a format that can be easily transferred to other statistical analysis packages.

MASTS

The result of this project is the Microcomputer-Assisted Study and Testing System (MASTS). MASTS is contained on two disks (Figure 1). The author disk contains the authoring program which creates the data and index files to be stored on the student disk. The student disk contains the student program which executes according to the data stored by the authoring program. Copies of the student disk are distributed to each student in a CMI course. The programs for collecting and manipulating student data from the individual student disks are included on the author disk. The heart of MASTS is a two-dimensional variable instructional algorithm (Figure 2) with parameters and branching or looping conditions set by the authoring program. This algorithm is assembled by the author program through interactions with the author. To assemble an algorithm, the author need only be familiar with the options available in the algorithm model (Figure 2), as the authoring program is menu-driven and all options at any point in an authoring session are displayed on the monitor.

AUTHORING SYSTEM

An instructor wishing to use MASTS receives an author disk, a student disk, and a short booklet outlining the options available in the instructional algorithm for creating a CMI module. After reading the booklet, the instructor plans the assignments and tests and determines the criterion levels to allow the student to progress or receive remedial assignments. The instructor or clerical assistant may now interact with the authoring program to create the CMI unit. A complete CMI unit may consist of numerous assignments and tests, but MASTS does not require that the complete unit be entered in a single authoring session. The smallest amount of information that may be entered during a session is one module containing an assignment, the test based on the assignment, and the data such as competency level for advancing, number of test questions to give, whether questions are to be given in random or fixed order, and what remedial strategy is to be taken upon the student's failure to reach the established competency level.

Each assignment/test module is given a two-number label X, Y, which determines its position within the instructional algorithm. X represents the main assignment/test. Y represents a remedial assignment/test for the main assignment-test X. For example, the first assignment in a MASTS unit would have the label 1,0. If the author-specified competency level were met for test 1,0, then assignment 2,0 would be made. If the competency level were not met, the MASTS would execute the author-specified remedial strategy. The student could then repeat assignment 1,0, be given a remedial assignment Test 1,1, or be blocked from further assignments pending a visit to the instructor, at which time the block can be removed from the student's disk. When a remedial test is passed, the student is advanced to the next main assignment. When a remedial test is failed, the options are the same as outlined for 1,0. Thus, a remedial assignment/test prescribed for failing 1,1, would be 1,2. Passing 1,2 would advance the student to 2,0. If the last remedial test for a main assignment is failed (X,N), the student is automatically blocked and told to see the instructor. When a student has successfully completed all of the main assignments on a disk, he is instructed to turn in the disk to the instructor. Any number of main assignments (X) and any number of remedial assignments (Y of X) may be created within the file boundaries of the microcomputer disk-operating system.

To begin a MASTS authoring session, the author places the author disk in the microcomputer and turns on the power. The author program automatically loads, runs, and presents the author with a master menu. The author may select one of several options. If he selects to create new assignment/test modules, he must put a student disk into a second disk drive, and is so informed by the author program. The author will then be asked to enter the X,Y number of the new module. After this information is entered, the program will prompt the author to enter the assignment, the text of which can be any length. At the end of the assignment, the author enters the letter "S" on a new line to signal the end of the assignment. At this point, the author may instruct the program to execute the test immediately after the assignment or allow the student to leave the computer to complete the assignment. It is also possible to run a program specified by the author at this point, which allows the integration of other CBI programs with MASTS.

The program will now prompt the author to enter the test. The test is constructed through interaction with a menu-drive subroutine which allows any number of (mixed) multiple choice, true-false, or key word questions to be entered. The menu allows the author to select a question type or end the test. The process for entering the text for all types of questions is similar to that for entering the assignments. A multiple choice question may have any number of choices since they will be presented one at a time to the student. (After all choices have been shown, they will be repeated until the student picks one.) The key word question allows the author to specify a number of key words or character strings to be considered a correct answer to the question. In the student program, a key word question asks the student to type in a response. This response will be searched by the program to determine if it contains any of the key words. If it does, it will be counted as correct. After entering a test question, the program returns to the menu which allows other questions or the end of the test to be specified. At the end of the test, the author is prompted to enter the competency level in percent for passing to the next main assignment. He is then prompted to enter his choice for a remedial strategy for students who do not pass. When this information has been entered, the program will return to the master menu. From the master menu, the author may add other modules as outlined above or print a master student disk when all modules for the MASTS unit have been entered. When the latter option is selected from the master menu, the program will prompt the author to enter the information fields he wishes to gather on each student's disk. The author simply types a one-line instruction, such as "Please enter your last name," for each information field he wants. Any number of fields may be entered, and they will be presented before the student's first assignment. The responses will be stored on each student's disk.

When the author finishes entering information fields, the author program will place this information on the student disk, along with other index information making the student disk complete. Instructions are then given to make copies of the master student disk for distribution, which necessitates the program returning to the microcomputer's disk-operating system.

STUDENT DISKS

Each student is given a copy of the master student disk. To use his disk, a student places it in any compatible microcomputer and turns the microcomputer on. The program automatically loads and starts at the proper point in the instructional algorithm. On the first session, the program gathers the information fields entered by the author and the first assignment is given. Then according to the instructions entered by the author: 1) the program stops and will not continue until the computer is turned on again, thus allowing the student to remove his disk and carry out the assignment before returning for a test; 2) the program proceeds directly to the test on the assignment, in which case, the assignment text could be used as an instructional frame rather than as an assignment as such; or 3) the program chains directly to another program which could be placed on the student disk by the author.

When a student finishes a test, he is given instructions which may be the next main assignment, a remedial assignment, the same assignment again, directions to see the instructor (a block is placed on re-entry until the instructor unlocks the disk), or instructions to turn the disk in, if the student has finished the last assignment. All scores for each test are stored in the studex file which also holds the information on a student's location in the algorithm and his re-entry status.

MASTS DATA COLLECTION AND DISPLAY

When students have completed their MASTS disks, the disks are turned in to the instructor for data retrieval. One of the options on the authoring program master menu is collecting data from disks. Each student disk is placed in the second disk drive and is read by the author program collect subroutine. The information fields and scores for each test are stored with those for all other students. When all student disks have been read, the instructor may use the result display option from the master menu. This option causes the computer to chain to a data display program that computes and displays statistical information for each test and each student. If a particular test has been repeated, the statistics are computed for each attempt. This information may be displayed on the monitor, printed on a line printer, or both. The student data is stored in standard ASCII text files so that simple programs may be written to transfer student data to other data base formats used by more sophisticated statistical analysis packages.

APPLICATION AND EVALUATION

MASTS has been used for two semesters in an upper-level media course for education majors at the University of Texas at Austin. Most of the class time for this course is spent teaching production processes while MASTS is used to assign outside readings describing media applications. In the past, assignments were made with a reading list, but many students did not finish the readings. Another approach used was to require students to write reaction papers or short reports based on the reading assignments, but this approach reduced the number of readings that could be assigned during a course. Using MASTS for these assignments allows the instructor to set due dates for completion that coincide with related in-class activities.

These first applications of MASTS have served as formative field evaluations helping to define needed changes and to catch bugs as only trial by fire can do. An added benefit from using MASTS has been that the computer anxieties of students, most of whom have never worked with a computer, have been lowered (according to pre- and post-MASTS questionnaires).

MASTS HARDWARE

A Radio Shack TRS-80 with 48K memory and two disk drive units is used for authoring and data collection from student disks. The student programs require only one disk drive and 32K memory.

CONCLUSIONS

The definite student enthusiasm and lack of major problems encountered in implementing MASTS indicate an open door for exploring the use of small scale CMI. MASTS or a similar system could form the heart of a programmed text or workbook. Another application might be in individual learning centers, using all forms of instructional materials as possible assignment media.

REFERENCES

Baker, Frank B. Computer Managed Instruction. Englewood Cliffs, N.J.: Educational Technology Publications, Inc., 1978.

Bork, Alfred. "Course Management System for Physics III." Proceedings of the Conference on Computers in the Undergraduate Curricula, 1977, 213-222.

Brockmeier, Richard. "Report on a Highly Used Computer Aid for the Professor in His Grade and Record Keeping Tasks." Proceedings of the Conference on Computers in the Undergraduate Curricula, 1977, 93-100.

Hilgendorf, Allen. A Study of the Transportability and Effectiveness of the Un-Stout CMIS and Individualized Instructional System Based Upon Learning Styles. Menomonie, Wisc.: Univ-Stout, 1976. (ERIC Document Reproduction Service No. ED 131 305).

McDonnell Douglas Corp. Advanced Instructional System. Brochure distributed at AECT Convention, 1979.

Nievergelt, Jurg, and others. Alternative Delivery Systems for the Computer-Assisted Instruction Study Management System (CAISMS). Urbana, Ill.: Illinois University, February 1978. (ERIC Document Reproduction Service No. ED 149 785).

Reigeluth, Charles M. TICCIT to the Future: Advances in Instructional Theory for CAI. Salt Lake City, Utah: Brigham Young University, 1978. (ERIC Document Reproduction Service No. ED 153 611).

Roll and Pasen. "CMI Produces Better Learning in an Introductory Psychology Course." Proceedings on the Conference on Computers in the Undergraduate Curricula, 1977, 229-238.

Van Matre, Nicholas H. "Problems in Researching Computer-Managed Instruction." Paper presented at the American Educational Research Association, Toronto, Canada, March 1978.

RIBYT -- A DATA BASE SYSTEM FOR FORMAL TESTING AND SELF-ASSESSMENT

F. Paul Fuhs

School of Business
Virginia Commonwealth University
1015 Floyd Ave.
Richmond, Va. 23284
804-257-1737

ABSTRACT

This paper describes the function and structure of a data base system called RIBYT, which simultaneously controls and associates questions in question pools for many courses of instruction. The data base stores questions created by both faculty and students and is used for formal testing and student self-assessment. The structure of the data base allows rapid retrieval of multiple types of sets based on predetermined logical associations. The data base also provides for the collection and association of feedback information not only on student performance, but also on the quality of the questions in the data base. This quality is assessed through students' subjective comments about the questions they receive and through statistical item analysis. Feedback is also provided to students on how they are able to improve the quality of their question input. The structure of the data base is easily transferable to many data base systems which currently exist.

INTRODUCTION

Many educators administer one or more pools of objective test questions as the basis for examinations. These pools usually are in the form of multiple-choice, fill-in-the-blank, or column-matching questions. Educators have a need to generate, organize, store, modify, and retrieve such test questions[1].

The logical associations among the questions within a pool present a difficult administrative problem. Questions must simultaneously be ordered in a number of ways; they may be grouped into sets by course topics, by textbooks, by page numbers within textbooks, by tests in which the questions were used, and by many other categories. Question pools are also difficult to manage because they are dynamic; questions have different life spans, subject to such factors as over-exposure in usage, changes in textbooks, and obsolescence due to technological changes. New questions must continually be added to these pools and each new question forms multiple associations with previously existing questions. In addition, periodic rewording and pruning of existing questions are required and, unless the questions in a pool are already well organized, it is difficult to detect synonymous questions.

Many traditional file systems have attempted to solve these associational problems. Yet, frequently these systems are expensive, overly complicated, and difficult to use. They lack the capability to make multiple associations among the data items.

DATA BASE TECHNOLOGY

Data base technology over the past ten years has been successfully applied to many storage and retrieval problems especially where multiple associations exist among data elements. Such associations are logically represented as relations (2), hierarchical structures (8), and simple or complex networks (9). They are physically implemented through such methods as hashing, tables, and lists. The data base approach has penetrated business and government and is in the process of replacing many traditional file systems. DBMS are currently available even on minicomputers like IBM's System/38, Hewlett Packard and DEC machines. However, educational institutions have been slow to use data base systems except for some purely administrative functions.

---

[1] Publishing companies like Harcourt, Brace, Jovanovich (7) and SRA (10) have recognized this need and are willing to supply such pools in certain courses to faculty who adopt their texts. Their pools, however, lack associational structure and are merely sequential files organized by text page number.

FUNCTIONS OF RIBYT

In this paper, we describe the functions and structure of the data base system called RIBYT at Virginia Commonwealth University. This system applies data base technology to the associational problems of using and maintaining pools of questions for formal testing and student self-assessment[2]. The single data base contains multiple pools of questions, one pool for each course using RIBYT. The questions are of various types, including multiple-choice questions with single or multiple correct answers; fill-in-the-blank questions with multiple and alternative correct answers; and column-matching questions. Each pool is under the direct control of the faculty members responsible for the course. In addition to faculty, students in some courses are allowed to create and add questions into the data base. Student questions, however, are considered unedited until reviewed by a faculty member. Such questions, while physically intermingled with edited questions, are logically distinct. During the review process, student questions are accepted, modified, or deleted[3]. The question pools grow at a much more rapid rate when students are allowed to enter questions. With a large pool of questions and the security provided by a data base system, faculty allow student self-testing in a nonthreatening spirit similar to the self-assessment procedures afforded members of ACM (11). Students view either faculty selected questions or random sets of questions, and receive feedback on their performance. The faculty control which questions the students are allowed to view, but students control (within limits) the number of questions, the course question topics, and text page ranges over which questions may be selected for self-assessment. Students are generally enthusiastic about participating in question generation because they understand the merits of the self-testing.

To improve the quality of student questions, faculty, while reviewing the questions, optionally enter into the data base comments directed to the student authors of the questions. This feedback is a learning experience in test construction, but even more so in course content. There are two other feedback mechanisms to improve the quality of the questions in the data base. First, from an inspection of patterns of student responses faculty may detect acceptable alternative responses that were not previously considered. Secondly, students can enter into the data base complaints concerning any of the self-assessment questions they have received. As a result faculty and students critique the self-assessment questions. Questions for formal examinations are selected from questions which have already passed through these filters.

Student responses to test questions are used not only for student performance evaluation, but also for statistical item analysis. Each response entered into the data base is associated with a unique question number within the data base, a course test number, the question number within the test, the student making the response, and the student's score for the response.

The RIBYT data base system associates two course topics with each question in the data base. Topics assigned to questions by students are considered temporary, until reviewed for editing by faculty. Each course can organize its permanent topics hierarchically and independently of any other course. There is no limit to the depth of any of the hierarchies. The topics for all courses are physically in the same data sets of the data base, but logically kept distinct. The hierarchical organization of topics provides flexible retrieval possibilities. Questions may be retrieved based on specified topics. Alternatively, they may also be retrieved by topics that exist as children nodes in the hierarchical topic structure. Retrieval, therefore, is made as generic or specific as desired. Faculty and student users can view at any time the hierarchical structure of the topics which, similar to the questions within any question pool, changes over time.

IMPLEMENTATION OF RIBYT

RIBYT was designed under the assumption that faculty and students using the system would know little or nothing about data base processing. Application programs written in COBOL and BASIC act as the communication between the DBMS and the users, who access the system through batch and on-line devices. RIBYT is implemented on a Hewlett Packard 3000 Series II computer using the DBMS package, IMAGE, which has been widely acclaimed (5). The design of RIBYT is not restricted to IMAGE, however. With minor modifications RIBYT can be implemented on many network-oriented DBMS. CINCOM's TOTAL DBMS is but one example (3).

---

[2] The self-assessment aspect of RIBYT is one of the data base's unique characteristics, from which it derives its name (Review It Before You Test).

[3] The review process is also used to modify or delete questions which have failed statistical item analysis.

ASSOCIATIONAL STRUCTURE OF RIBYT

Figures 1 and 2 illustrate the 22 data sets (files) of RIBYT. There are eight master[4] and 14 detail data sets. Master data sets contain records stored by hashing. Each master record, composed of one or more data items, can be linked to one or more detail data sets. Master data sets can be viewed as either independent files or as sets of records where each record acts as a chain head to one or more records in a detail data set. Within a detail data set records are linked by forward and backward pointers to form sets. A detail record may belong to one or more master data sets. Master data sets are of two types, manual and automatic. Unlike records in manual data sets, automatic data set records are stored or deleted by the DBMS itself. These records act as chain heads for records in detail data sets. With this data base architecture, networks of associations among records and data sets can be created and then manipulated by the DBMS. Figures 1 and 2 show for each data set its name, its type[5], and its associations with other data sets. Within a detail data set a chain of records can be further logically ordered by specifying one of the data items within the record as a sort item[6].

Before faculty or students store or retrieve questions, information concerning the users is stored in the data set called ROSTER[7], course information in the data set TEXTBOOK-MASTER, and system initialization information in the data set LAST-MASTER. Each record in ROSTER contains a user's name, course and section number, a status (faculty or student), and a unique user identification number. TEXTBOOK-MASTER contains one record for each block of twenty pages of each textbook. Each record contains the textbook title, and a data item BOOK-BLOCK composed of course and an associated textbook number and the page block. Such structuring gives more rapid retrieval for those questions based on textbook page numbers. The data set LAST-MASTER contains one record. Stored here are the sequence numbers of the last question entered into the data base (LAST-QUES), the last temporary topic (LAST-TEMP-TOPIC), the last permanent topic (LAST-PERM-TOPIC), and a faculty password (FAC-CODE). The latter allows faculty to perform such special functions as dumping an entire question pool for a course. In addition to the security provided by the operating system and the DBMS, FAC-CODE and ROSTER are used to control access to the data base. SYSTEM-STACK is the other system data set, implemented as a stack that contains information on the last 50 interactive or batch jobs run. Each record contains a time stamp, plus a user and program identification number. This information aids in backing out of system malfunctions and detecting unauthorized access.

A user, wishing to enter questions into the data base, begins a session at a CRT by choosing from a menu of textbooks associated with the user's courses. This menu uses information from the data sets ROSTER, ALTERNATIVE-COURSE, and TEXTBOOK-MASTER. Then, for each question, users are prompted for background information on the question before entering the text of the question. This information will be entered into the data set QUESTION-INDEX. The user enters two course topic numbers, a textbook page number, and the type of question. The application program assigns a permanent or a temporary number to each new topic. The names and numbers of new topics are entered into the data set TOPIC-MASTER. If the user wishes to index a question on only a single topic, the second topic becomes a dummy. The program then adds to the record in QUESTION-INDEX the DATE, the BOOK-BLOCK, a unique question number (QUES-NUM), the version number of the question (QUES-VERS), and the user's identification number. The status of the question (QUES-STATUS) is set to "edited" if the user is a faculty member, otherwise it is set to "unedited." The other data items in QUESTION-INDEX are initialized to zero.

---

[4] Aside from the terms "master" and "detail" which are peculiar to Hewlett Packard, we will adhere to CODASYL(1) terminology as closely as possible.

[5] In figures 1 and 2, B=Binary and X=Alphanumeric. The numeric value following the alphanumeric specification "X" represents the number of bytes assigned to that data item.

[6] In figures 1 and 2 these sort items are designated as "S" under the appropriate data items.

[7] For readability the data base, data sets, and data items are shown in capital letters.

After all data item values have been constructed for a record in QUESTION-INDEX, as the record is physically stored, the DBMS creates multiple associations using hashing and linked lists. The question's background information in QUESTION-INDEX is linked to its author in ROSTER, the textbook referenced in TEXTBOOK-MASTER, the question's topics in TOPIC-HEAD, and its unique question number in QUESTION-HEAD.

The text for a question and its answers are next entered into the data sets QUESTIONS and ANSWER-CHOICES. For fill-in-the-blank questions the text is stored in QUESTIONS and the answers are stored in ANSWER-CHOICES. Alternatively acceptable answers for each blank are stored in ANSWER-CHOICES. For multiple-choice questions, the stem of the question is stored in QUESTION and the choices in ANSWER-CHOICES. Each choice is designated as correct or wrong (CORRECT-WRONG). One can not afford to assign to each question, choice, and answer a fixed length equal to the largest possible number of bytes the text might need, rather, data base technology is used to solve this problem. Text is considered as 40 or 72 character blocks linked together. Records in QUES-VERS-HEAD are chain heads for sets of character blocks in the data sets QUESTIONS and ANSWER-CHOICES. Within the data set QUESTIONS each textual block of characters is assigned a sequential line number (LINE-NUM), which is used as a sort item to keep the text in correct logical order on a chain[8]. Since any choice of a multiple-choice question may also have more than one line of text, the sort item called SEQUENCE is composed of the alphabetic choice designator (A,B,C,D,or E) concatenated to a text line number within the choice. This system automatically keeps the choices and the lines of text within each choice in logical order. The total number of times each choice is selected as an answer in either formal testing or self-testing is later stored in the data item RESP-TOTAL.

QUESTION RETRIEVAL

There are many selection options for question retrieval from RIBYT, aside from retrieval for editing purposes. Question retrieval is first divided into formal testing and self-testing. Faculty can restrict questions so that these are used only in formal testing. Each type of testing is further divided into three options. Under the first option faculty specify by question number the actual questions to be included in a test. Under the second option faculty specify that a single set of questions are to be randomly selected for an entire class. Under the third option faculty indicate that a different set of questions are to be randomly selected for each student. Faculty may specify that questions are to be selected based on one or more topics, page ranges, or any combination of topics and page ranges. In self-testing, faculty may leave topic and page selection to each student's choice.

DATA SETS USED IN QUESTION RETRIEVAL

Three data sets are used to control question retrieval. These data sets contain specifications that are used by a retrieval program to select questions from the data base for formal testing and self-testing. The first data set, TESTS, has a primary key COURSE-TEST-NUM, which is a combination of the course designation and a test number within the course. Each record in TESTS represents a single test. Each record has a SECURITY-CODE to control access to the test. The percentage that each test represents towards the total course grade is stored in TEST-WEIGHT. The number of questions per test, the number of times a test may be taken, the average difficulty factor per test question, the types of questions, the selection options, and the mix between validated and unvalidated questions are stored in the data set TESTS. When faculty wish to designate the actual test questions, the set of question numbers is stored in the data set SPECIFIED-QUES by question number (VERS-NUM). Each number acts as a search argument to retrieve the text of one question. This retrieval is performed as follows. First the system uses the value VERS-NUM in SPECIFIED-QUES to hash into the data set QUES-VERS-HEAD. The retrieved record then points to sets of textual blocks, representing a question, its choices, and answers in the data sets QUESTIONS and ANSWER-CHOICES.

---

[8] Records in any of the data sets are not necessarily stored in the same order in which they were input.

The data set PAGES-TOPICS is a multipurpose data set. Each record in PAGES-TOPICS stores either a textbook page range or one or two course topic identification numbers. Each record is linked to an individual test in the data set TESTS. The value of the data item PAGE-TOPIC-SWITCH indicates whether a record contains page or topic information. For pages, the data items START and END store page ranges and the data item BOOK-HIER stores a course textbook identification number. For topics, two topic identification numbers can be stored per record; one in START and one in END. BOOK-HIER is then used as a binary switch to indicate whether only designated topics or their hierarchical children are to be included in question retrieval. Question retrieval based on textbook pages is accomplished by hashing into TEXTBOOK-MASTER and following appropriate chains into the data set QUESTION-INDEX and from there using the combined data items, QUES-NUM and QUES-VERS, as a key to hash into QUES-VERS-HEAD. Then the program follows chains into QUESTIONS and ANSWER-CHOICES as described above. Question retrieval by topics is similar after entry into QUESTION-INDEX from TOPIC-HEAD. Since all hierarchical twins of a given parent topic are organized as a set in the data set TOPIC-INDEX, whose members are linked to a topic in TOPIC-HEAD, sub-topics are found by first hashing into TOPIC-HEAD and following a chain into TOPIC-INDEX.

STORING AND RETRIEVING TEST RESPONSES AND TEST SCORES

Information concerning student responses is stored in six data sets; TEST-SCORES, TESTS, RESPONSES, STUDENT-GRIPES, ANSWER-CHOICES, and RESP-FEEDBACK. The set of all test scores for a given test is linked together within the data set TEST-SCORES, and this set is itself linked to the data set TESTS. Within the data set TEST-SCORES is a chain associating all tests for each student. Since RIBYT handles the possibility that the same student takes the same test more than once, each test score is made unique by being associated with a time stamp DATE-TIME in TEST-SCORES.

The student responses to each test question are stored in the data set RESPONSES. Each record is composed of a question identification number (VERS-NUM), the question number within the test, the text line number within the response, a code indicating whether the response was correct or incorrect, the student's response, and a combined data item (USER-DATE-TIME) formed from the student identification number and the time stamp. Each response in the data set RESPONSES is on two link paths, one originating from the data set QUES-VERS-HEAD, the other from the data set TEST-HEAD. In addition, since it is desirable to have the responses continually sorted by test question number within a test and since there may be more than one text block per student response, the two sort items, TEST-QUES-NUM and LINE-NUM, are used as major and minor sort fields for the records on a USER-DATE-TIME chain.

Besides receiving the traditional performance feedback to their responses (a list of correct answers, questions, and test scores), students may receive textual comments concerning particular responses. Faculty store these in the data set RESP-FEEDBACK. After students receive feedback on their tests, they may enter into the data set STUDENT-GRIPES their reactions to any individual test questions, anonymously, or by student identification number. Course grades can easily be determined based on the data sets TESTS and TEST-SCORES.

STATISTICAL ITEM ANALYSIS

Two measures of a question's quality under statistical item analysis are item difficulty and item discrimination (4). A difficulty index is calculated concurrently for every question in a test. The set of student responses is retrieved by hashing into the data set TESTS with the key COURSE-TEST-NUM and obtaining all chained records in the data set TEST-SCORES. For each record retrieved the data item values of USER-NUM and DATE-TIME are used as a concatenated key to hash into the data set TEST-HEAD. All test responses for a single student are retrieved from the data item CORRECT-WRONG of the data set RESPONSES through a chain originating in the retrieved TEST-HEAD record, and this procedure is then repeated for each student. The retrieval is rapid and passes over all other tests in the data base. The calculated difficulty indices are then stored in the data set QUESTION-INDEX. The calculations for statistical item discrimination require the same data sets as item difficulty determination.

CONCLUSION

The associational power inherent in the data base system RIBYT allows many relationships to be made and maintained among courses, textbooks, course topics, students, faculty, tests, test questions, test construction options, and student responses to questions. RIBYT stores subjective and statistical feedback on the quality of the questions in the data base and associates this feedback with the test questions. This system supports formal testing and self-assessment procedures and allows the storage of many types of questions, while imposing no limit on the size of any question or answer. The centralization of the logical associations into one data base provides better security and easier maintenance than traditional file systems.

ACKNOWLEDGEMENTS

I am grateful for the assistance of Dr. Kathleen Corrigan Fuhs and Mr. Paul Thompson for helping to specify and clarify some of the user requirements of RIBYT.

REFERENCES

(1) CODASYL, CODASYL Data Base Task Group Report, ACM, New York, 1971.

(2) Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," CACM, 13,6, 1979.

(3) Datapro, TOTAL, Delran, N.J.: Datapro Research Corp., March 1978, section M12-132-101 to M12-132-104.

(4) Downie, N.M. and R. W. Heath, Basic Statistical Methods, 2nd. ed., New York: Harper and Row, 1965, p.228.

(5) Gepner, Herbert I., "User Ratings of Software Packages," Datamation (December 1978), 183-185.

(6) Hewlett Packard, Image, Data Base Management System Reference Manual, Santa Clara, Ca.: Hewlett Packard, 1978.

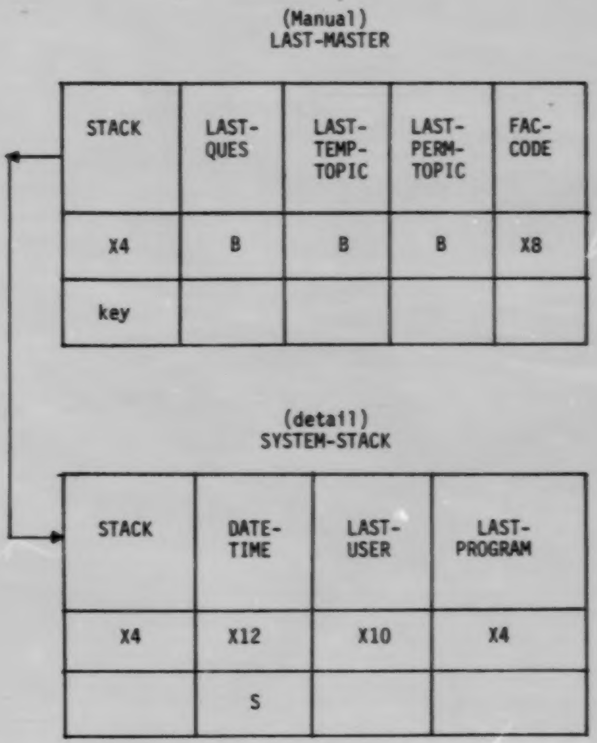
(7) Hilgard, E. R., R.L. Atkinson, and R. C. Atkinson, Introduction to Psychology, 7th ed., New York: Harcourt, Brace, and Jovanovich, 1979.

(8) IBM, IMS/VS Application Programming Reference Manual, order no. SH 20-9026, White Plains, New York, 1978.

(9) Kroenke, David, Data Base Processing, Chicago: SRA, 1977.

(10) SRA, "Mid-Term Blues? Computerized Test Service Available," Data Processing News (Spring 1979),4.

(11) Wong, J. W. and G. Scott Graham, "Self-Assessment Procedure VI," CACM, 22,8 (August 1979),449-454.



(Manual)
LAST-MASTER

STACK	LAST-QUES	LAST-TEMP-TOPIC	LAST-PERM-TOPIC	FAC-CODE
X4	B	B	B	X8
key				

(detail)
SYSTEM-STACK

STACK	DATE-TIME	LAST-USER	LAST-PROGRAM
X4	X12	X10	X4
	S		

FIGURE 1: SYSTEM CONTROL DATA SETS OF RIBYT

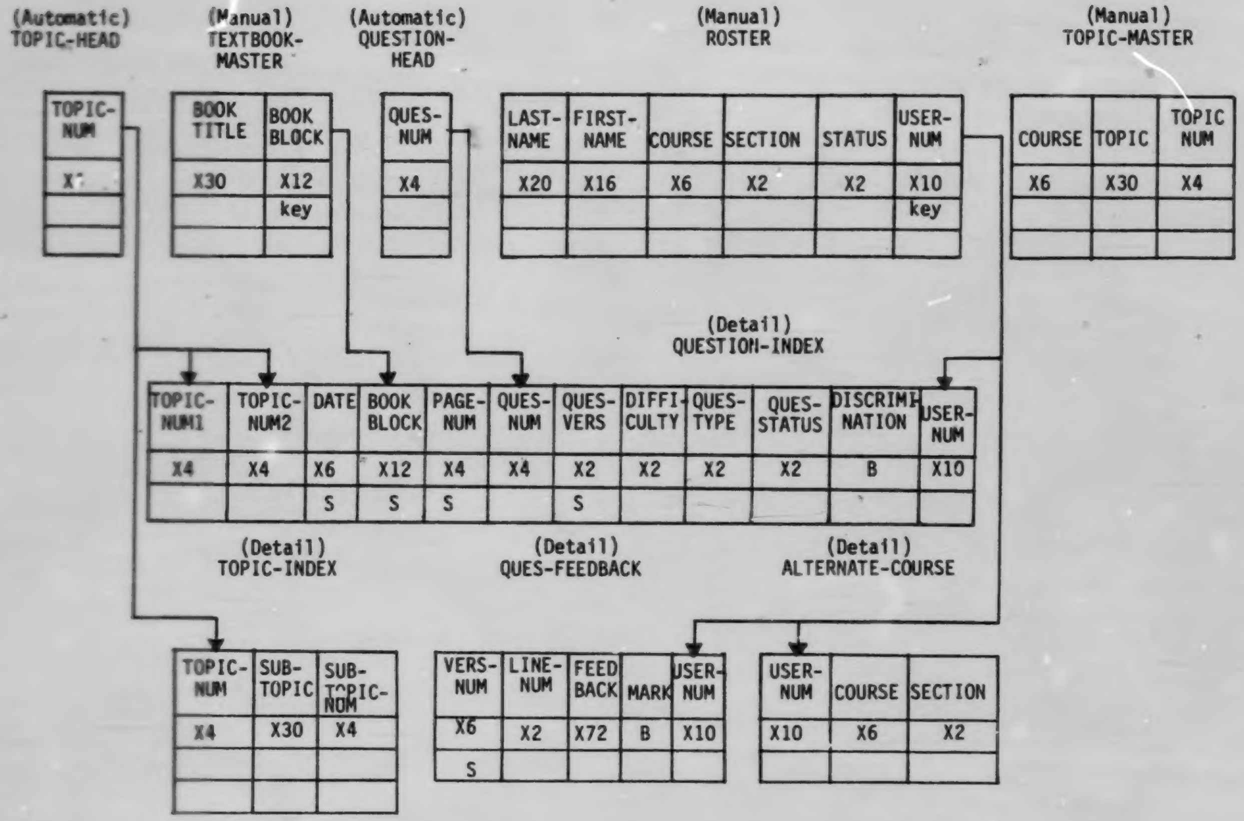

(Automatic)
TOPIC-HEAD
TOPIC-NUM
(Manual)
TEXTBOOK-MASTER
BOOK TITLE
BOOK BLOCK
X30
X12
key
(Automatic)
QUESTION-HEAD
QUES-NUM
X4
(Manual)
ROSTER
LAST-NAME
FIRST-NAME
COURSE
SECTION
STATUS
USER-NUM
X20
X16
X6
X2
X2
X10
key
(Manual)
TOPIC-MASTER
COURSE
TOPIC
TOPIC NUM
X6
X30
X4
(Detail)
QUESTION-INDEX
TOPIC-NUM1
TOPIC-NUM2
DATE
BOOK BLOCK
PAGE-NUM
QUES-NUM
QUES-VERS
DIFFICULTY
QUES-TYPE
QUES-STATUS
DISCRIMINATION
USER-NUM
X4
X4
X6
X12
X4
X4
X2
X2
X2
X2
B
X10
S
S
S
S
(Detail)
TOPIC-INDEX
TOPIC-NUM
SUB-TOPIC
SUB-TOPIC-NUM
X4
X30
X4
(Detail)
QUES-FEEDBACK
VERS-NUM
LINE-NUM
FEED BACK
MARK
USER-NUM
X6
X2
X72
B
X10
S
(Detail)
ALTERNATE-COURSE
USER-NUM
COURSE
SECTION
X10
X6
X2

FIGURE 2: DATA SETS OF RIBYT

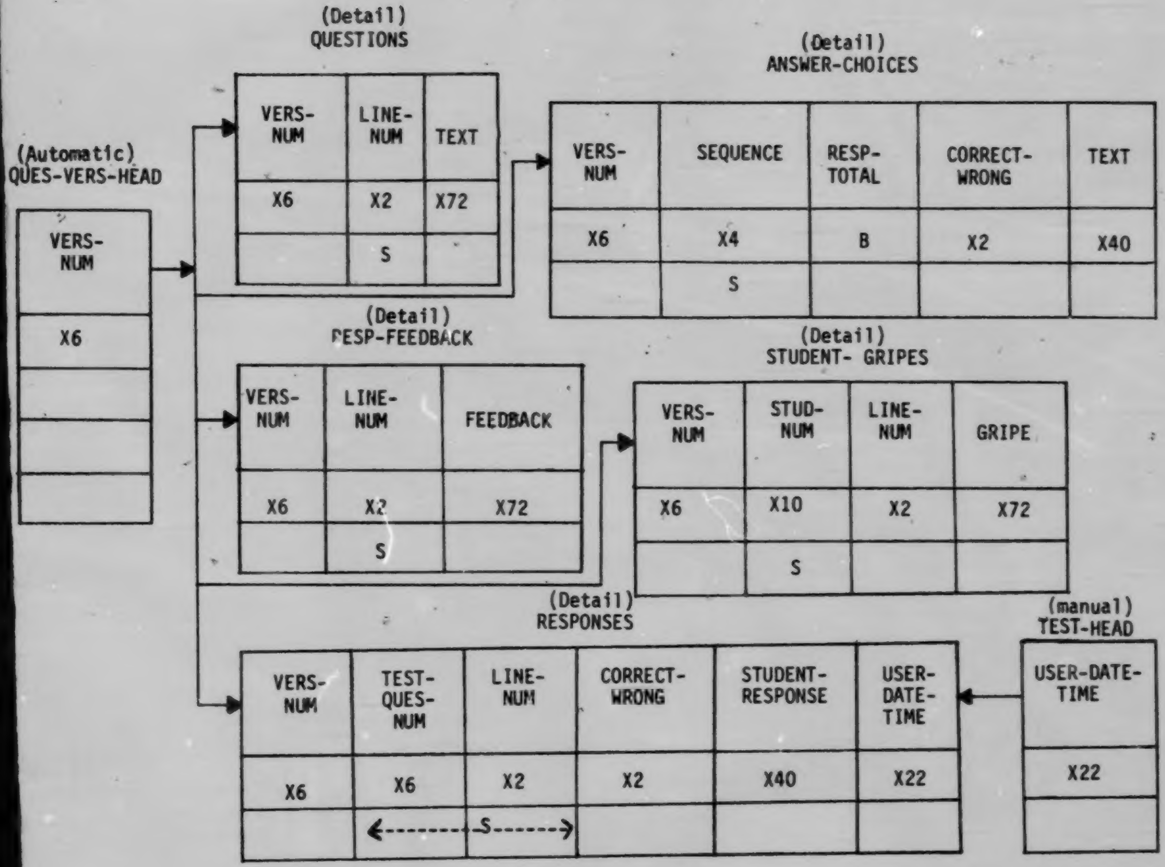
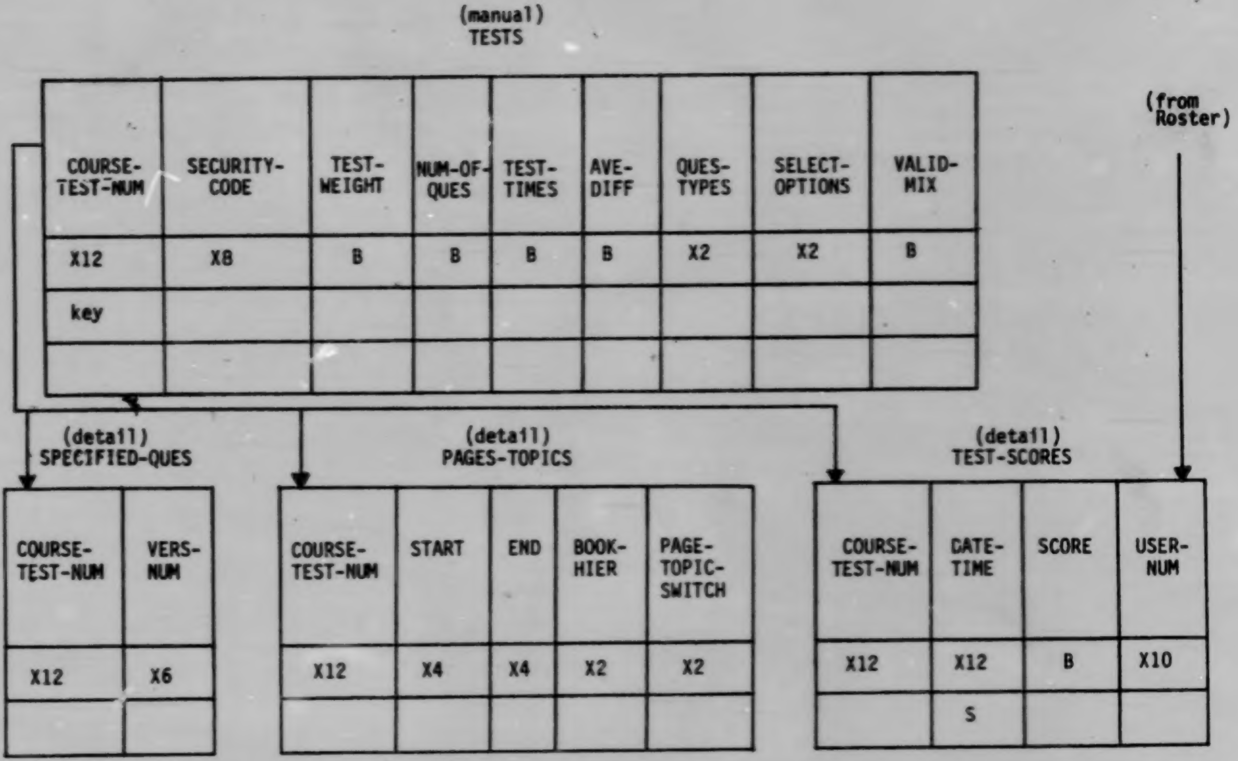

(Detail)
QUESTIONS
VERS-NUM
LINE-NUM
TEXT
X6
X2
X72
S
(Detail)
ANSWER-CHOICES
VERS-NUM
SEQUENCE
RESP-TOTAL
CORRECT-WRONG
TEXT
X6
X4
B
X2
X40
S
(Automatic)
QUES-VERS-HEAD
VERS-NUM
X6
(Detail)
RESP-FEEDBACK
VERS-NUM
LINE-NUM
FEEDBACK
X6
X2
X72
S
(Detail)
STUDENT- GRIPES
VERS-NUM
STUD-NUM
LINE-NUM
GRIPE
X6
X10
X2
X72
S
(Detail)
RESPONSES
VERS-NUM
TEST-QUES-NUM
LINE-NUM
CORRECT-WRONG
STUDENT-RESPONSE
USER-DATE-TIME
X6
X6
X2
X2
X40
X22
S
(manual)
TEST-HEAD
USER-DATE-TIME
X22

Figure 2 (cont): DATA SETS OF RIBYT



(manual)
TESTS

COURSE-TEST-NUM	SECURITY-CODE	TEST-WEIGHT	NUM-OF-QUES	TEST-TIMES	AVE-DIFF	QUES-TYPES	SELECT-OPTIONS	VALID-MIX
X12	X8	B	B	B	B	X2	X2	B
key								

(detail)
SPECIFIED-QUES

COURSE-TEST-NUM	VERS-NUM
X12	X6

(detail)
PAGES-TOPICS

COURSE-TEST-NUM	START	END	BOOK-HIER	PAGE-TOPIC-SWITCH
X12	X4	X4	X2	X2

(from Roster)

(detail)
TEST-SCORES

COURSE-TEST-NUM	DATE-TIME	SCORE	USER-NUM
X12	X12	B	X10
	S		

Figure 2 (cont): DATA SETS OF RIBYT

COMPUTER MANAGED PLACEMENT
IN
MATHEMATICS INSTRUCTION
FOR
HEALTH OCCUPATIONS STUDENTS

Thomas A. Boyle
Purdue University
West Lafayette, Indiana 47907
(317) 749-2256

Peter T. Magnant
Indiana Vocational Technical College
1315 East Washington Street
Indianapolis, Indiana 46202

INTRODUCTION

The Indiana Vocational Technical College at Indianapolis (IVTCIN) maintains an instructional program to serve the mathematics needs of students beginning study in health occupations technologies. The program is based on Streeter and Alexander's Fundamentals of Arithmetic (1), beginning with whole number operations, continuing with fractions, decimals, and percents, and extending to ratios and signed numbers. The materials are organized in 15 modules, each with a pre-test, a practice test, and a mastery test used to guide individual students to the study materials appropriate to their evident needs.

Although the instructional materials in the program proved appropriate and were reasonably well received by students using them, certain problems soon developed in guiding students to the modules specific to their needs. In particular, the following were noted:

1. The handling of pre-tests on an individual basis was time consuming and otherwise a burden on testing personnel.
2. Because of the limited number of pre-test forms and the level of supervision for actual testing, it was possible for students to use the pretest materials in ways which would fool themselves regarding their mathematics skills.
3. Capable students were bothered by the need for working through several tests in order either to find a suitable starting module or to demonstrate competency and by-pass the program entirely.
4. Neither the student nor the staff could readily get an impression of how the student stood with respect to the whole program.

COMPUTER APPLICATION

In an effort to mitigate these problems, work was begun early in 1978 to adapt test materials and procedures which had been in use at another IVTC institution (2). These materials consist of a programmed test format, which require students to attempt equal numbers of test items in each of several categories, together with computer scoring and data processing. This use of a computer enables first the scoring of each student in each item category, with subsequent determination of score statistics for each student group. As will be shown in subsequent examples, this use of a computer yields information in detail which would be practically impossible to get from hand scoring. Previous work with the combination of test format and computer data processing had demonstrated efficiency in the use of test administration time and the possibility of rendering information from each of several relatively independent item categories (3).

Figure 1 shows the computer scoring output for a select group of students. The category of subtest scores appear in six columns following the student names. A key at the bottom serves to identify six subtests. For this test some grouping of modules was necessary, e.g., instructional modules 1-3 dealing with whole numbers are all represented in subtest one. The key further identifies the SCR column as

```
I V T C INDIANAPOLIS 4/6/79

        S U B T E S T   S C O R E S
 NAME  1    2    3    4    5    6   1-6    NF                ERROR ITEMS

Bob W. 3   -4    2    1   -4    1   -1    37 51 32 18 50  4  5 14 26 61 72 76 82 71 79 8
       .43 -.57  .33  .17 -.80  .17 -.03

Bev H. 5   -2   -2   -5    1   -4   -7    44 13 33 53 17 45 52 39 58 14 66 76 82 71 85 7
       .83 -.22 -.29 -.63  .14 -.57 -.16

Sue D. 7   -4   -2    0   -1   -2   -2    48  7 24 21 53 28 18 50 57 40 14 66 76 82 71 6
      1.00 -.40 -.29    0 -.13 -.22 -.04

Jo B.  7   -2    1   -4    0   -2    0    48 31 42 54 50 45 23 38 59 44 66 76 71 79 62 6
      1.00 -.20 -.13 -.57    0 -.25    0

Pat W. 7    5    3    5    4   -1   23    42 18 56 52  5 59 19 66 76 81 74 62 83 80 78
      1.00  .56  .43  .83  .67 -.14  .55

Pam W. 7    8    2    0   -2    0   15    48 31 23 58 30 26 83 80 77 85 81 74 66 76 65 7
      1.00  .89  .22    0 -.25    0  .31

Joy D. 8    7    6   -1    2    2   24    48  7 24 52 30 26 67 77 85 75*77 79*73 65*79 6
      1.00  .78 1.00 -.13  .29  .20  .50

Dee R. 8    8    5   -1    5    0   25    46 53 56 52 14 72 76 77 85 81 70 65 66 76 65 6
      1.00  .89  .71 -.14  .71    0  .54

Liz K. 7    9    6   -2    7    8   35    46 13 43 42 14 79 66 85
      1.00 1.00  .86 -.25 1.00 1.00  .76

Kay B. 7   10    7   -7    8    8   33    48 13 43 42 57 30 79 85 65
      1.00 1.00 1.00 -.88 1.00 1.00  .69

                                  S U B T E S T   K E Y
**  1 = WHOLE NUMBERS ** 2 = FRACTIONS ** 3 =DECIMALS ** 4 = PERCENT ** 5 = RATIO **
    ** 6 = SIGNED NUMBERS **
** SCR = SUM OF SUBTEST SCORES ** NF = NUMBER OF ITEMS ATTEMPTED ** RATIO SCORE = SCORE
    DIVIDED BY RELATED NF **
```

Figure 1. Facsimile M-31 Mathematics Scoring Program Output

holding the sum of six subtest and the NF column as showing the total number of items attempted by each student. Numbers at the far right identify the first 15 mistakes made by each student.

TEST SCORING

Subtest scores for each student are presented in two numbers. The first, the raw score for each subtest, is based on one point for each correct response. A penalty is applied for making errors in sequence; this scoring yields an expected score of zero for a student strategy of pure guessing. If performance is worse than that from pure guessing, i.e., if a student consistently makes errors in a subtest, then that subtest score becomes negative. The second number, the ratio score appearing below each raw score, is the result obtained from dividing the raw score by the number of related items attempted. As should be evident, the computation of seven raw scores and seven ratio scores would present a formidable task for anyone attempting hand scoring of this type of test.

APPROPRIATE PLACEMENT

The students represented in Figure 1 were selected to present a range of student performance. Bob, the first student, obtained a row of scores which could well result from guessing. It appears likely that he made consistent errors in items dealing with fractions and ratios, but he earned only three points in seven attempts with whole-numbers, so we may be reasonably confident Bob should start at the beginning of the instructional program.

```
                     INDIANA VOCATIONAL TECHNICAL COLLEGE
                                 INDIANAPOLIS
                   M - 31 MATHEMATICS SKILLS ASSESSMENT PROGRAM
                                     FOR
Pat W.                                                    4 679    80981    44
     THIS PROGRAM IS AIMED AT HELPING YOU DEVELOP THE BASIC MATHEMATICAL SKILLS NEEDED
TO DO AN EFFECTIVE JOB IN THE IVTC SPECIALTY YOU INTEND TO ENTER.  THE PROGRAM HELPS YOU
AND YOUR INSTRUCTOR FIRST BY NOTING THE PARTS OF THE IVTC MATHEMATICS SEQUENCE IN WHICH
YOU MAY ALREADY HAVE ADEQUATE SKILL.  NOTE IS ALSO MADE OF THE SECTIONS, OR MODULES IN
WHICH YOU APPEAR TO NEED ADDITIONAL HELP AND PRACTICE.  AS YOU KNOW, MOST PERSONS EN-
TERING TECHNICAL EDUCATION NEED TO IMPROVE THEIR SKILLS IN SOME PARTS OF MATHEMATICS.

     THE PRIMARY PURPOSE OF THIS ASSESSMENT PROGRAM IS TO HELP YOU GET STARTED AT THE
POINT WHERE YOU CAN DO YOURSELF THE MOST GOOD.  ONCE YOU KNOW THE MATHEMATICAL MODULES
YOU NEED TO WORK ON, YOU WILL FIND YOUR INSTRUCTOR AND THE LEARNING LABORATORY STAFF
READY TO HELP.  AND AFTER A PERIOD OF INSTRUCTION AND PRACTICE, YOU CAN CALL ON THE
SKILLS ASSESSMENT PROGRAM TO CONFIRM THE PROGRESS YOU HAVE MADE.

     THE FOLLOWING MESSAGES SHOULD HELP -

Pat W.
     YOU WORK SOMEWHAT FASTER THAN THE AVERAGE WHEN TAKING THE TEST.  THIS IS A REASON-
ABLE STRATEGY, BUT YOU MAY BE GUESSING ANSWERS TO ITEMS WHICH YOU COULD SOLVE IF YOU
SPENT A LITTLE MORE TIME ON THEM.

     IT APPEARS YOU DO YOUR BEST WORK WITH THE MATHEMATICS IN MODULES 1 - 3.  YOUR SCORE
OF 100 INDICATES ADEQUATE SKILL WITH WHOLE NUMBERS.

     JUDGING FROM YOUR RESPONSES, IT APPEARS LIKELY THAT YOU HAVE ADEQUATE SKILLS IN THE
FOLLOWING TOPICS.

                    TOPIC        SCORE       IVTC MATH MODULES

                   PERCENTS       83              TEN

     ALTHOUGH YOUR SCORES DO INDICATE SUFFICIENT SKILL, THERE MAY BE SOME ROOM FOR IM-
PROVEMENT, AND YOU MAY BENEFIT FROM A QUICK REVIEW OF ONE OR MORE OF THE MODULES LISTED.

     EVIDENTLY YOU HAVE MADE SOME PROGRESS IN LEARNING ABOUT FRACTIONS, HOWEVER YOU DO
APPEAR TO NEED MORE PRACTICE WITH THE MATERIALS IN MODULES 4 - 7.

     PROBABLY YOU CAN DO YOURSELF A LOT OF GOOD BY GETTING TO WORK ON THE MATERIALS IN
MODULES 8 - 9, EVIDENTLY YOU DO NOT HANDLE DECIMALS VERY WELL.

NO DOUBT, YOU REALIZE THAT YOU KNOW SOMETHINGS ABOUT RATIOS, BUT YOU DO MISS SOME POINTS
HERE AND PROBABLY WILL BENEFIT FROM ATTENTION TO IVTC MATH MODULE  ELEVEN.

EVIDENTLY YOU MADE MISTAKES ON THE FOLLOWING TEST ITEMS

                    18 56 52  5 59 19 66 76 81 74 62 83 80 78
```

Figure 2. Facisimile M-31 Mathematics Scoring Program Output

The next three students received very low scores in most subtests, yet scores which are evidently adequate in the whole number subtest. Bev had the lowest total score (-7) observed to date, but did five out of six whole-number items correctly. Evidently these students will benefit from starting with the instructional materials on fractions. Pat will also benefit from starting with the instructional materials on fractions; however, she obtained acceptable scores in other subtests.

Pam, Dee, and Joy, the next students listed, had scores characteristic of adequate performance in two or three subtests. Recommendations for these students would lead to intermediate modules in the instructional program. The scores for Pam and Dee indicate some study is needed beginning with module eight. Joy may well be advised to begin with module ten.

The two students remaining, Liz and Kay, received practically perfect scores

5

in all but one subtest. Evidently these girls made all but one of their mistakes on items in the percent subtest. Kay's scores are especially noteworthy. She worked rapidly on the test, attempting 48 items in the 30 minutes allowed. She got all items correct in five subtests yet made errors in all eight of the percent items she tried; the negative seven score results because the first error is not penalized.

INDIVIDUAL OUTPUT

The computer program which produces the test scores has been extended to yield a one-page message for each student on which the student's ratio scores are ranked, and the subtests on which he did best are identified. Appropriate messages are selected, including recommendations for action. Duplicate copies are produced for the student's file and for instructional supervisors. A facsimile of this output is presented as Figure 2.

SCORE STATISTICS

During the period to July 1979, the M-31 test was administered to approximately 1000 IVTCIN students. From the total group, some 763 were identified by instructional program. The response data from these students were grouped and test score statistics obtained for students entering each of seven health occupations programs at IVTCIN. The numbers of students identified by instructional program ranged from 31 for health career preparation to 455 entering the practical nursing program. Numbers of students in groups may appear to change and the sums of groups may be different from the total tabulated in Figure 3. This is because additional data were received and processed during the time the report was in preparation.

The score statistics were arranged by ranking the mean total ratio scores for the instructional groups. This

INDIANA VOCATIONAL TECHNICAL COLLEGE
INDIANAPOLIS
M-31 MATHEMATICS TEST SCORE SUMMARY
AUGUST, 1979

		INSTRUCTIONAL PROGRAM		WHOLE NOS.	COMMON FRACTIONS	DECIMALS	PERCENT	RATIOS	SIGNED NOS.	TOTAL SCORE	ITEMS ATTEMPTED
1	48	RESPIR THERAPY	MEAN	.93	75	.74	.49	.76	.71	.73	40.56
		N= 71	STD DEV	.09	.35	.33	.44	.39	.34	.22	8.62
2	28	MED LAB TECH	MEAN	.93	.72	.75	.45	.72	.70	.71	40.07
		N= 61	STD DEV	.15	.41	.29	.52	.37	.37	.25	9.06
3	46	RADIOLOGIC TECH	MEAN	.92	.66	.69	.47	.74	.66	.69	39.31
		N= 57	STD DEV	.13	.36	.34	.44	.39	.33	.21	7.84
4	42	SURGICAL TECH	MEAN	.95	.57	.68	.39	.67	.69	.65	39.25
		N= 57	STD DEV	.09	.48	.34	.46	.51	.39	.25	8.27
5	44	LIC PRACT NURSE	MEAN	.92	.62	.66	.27	.59	.49	.60	38.65
		N- 455	STD DEV	.15	.41	.30	.48	.47	.44	.25	8.42
6	37	MEDICAL ASST	MEAN	.93	.62	.66	.22	.45	.46	.56	35.64
		N= 39	STD DEV	.12	.42	.32	.50	.53	.52	.26	8.27
7	93	HLTH CAREER PREP	MEAN	.78	.33	.33	-.06	.21	.30	.31	37.45
		N= 31	STD DEV	.27	.54	.40	.60	.53	.50	.38	10.87
8		ALL GROUPS	MEAN	.92	.62	.66	.32	.61	.55	.61	38.52
		N= 810	STD DEV	.15	.42	.32	.50	.47	.44	.26	8.69

FIGURE 3. M-31 SCORE STATISTICS DISTRIBUTED BY INSTRUCTIONAL GROUP

ranking placed the respiratory theraphy students highest, and the health career preparation students lowest. The complete statistics for all seven student groups are presented in Figure 3. Mean scores are presented for all six of the M-31 subtests, as well as for total score and number of items attempted. With the exception of the items-attempted mean, the scores represented are ratio scores (i.e., the raw score divided by the number of related items attempted) for each student. As can be seen the subtest means range from 0.95, corresponding to a mean of ninety-five percent, down to a negative 0.06. This latter score indicates average test performance just slightly worse than would be expected for a student strategy of pure guessing.

Figure 3 indicates that, on the average, the M-31 test is of reasonable difficulty for students entering health service instruction programs at IVTCIN. However, the difficulty is not even for different groups. The respiratory therapy and medical laboratory technician trainees and the radiologic and surgical technicians find the test quite easy. The statistics show that, especially in the first groups, many students handle the designated range of arithmetic skill adequately. Unquestionably, there are many of these students who have little to gain from further study of the skills represented. Among the students who do seem to need further related study, the need appears to be greatest for percent drills.

Data for all students, including some tested after mid-July, show that there is some correspondence between the arithmetic abilities that have been acquired by the entering students and the sequence of instructional modules. On average, all groups score highest on the whole number subtest. With the exception of the percent scores, there is a trend toward lower scores in the subtests related to later instructional modules. The break in the sequence of scores, at the percent subtest, may indicate either need for special effort if students are to attain 80 percent performance or simply that the test items were unreasonably difficult. The break does call attention to an advantage of the test format: evidently some students who have difficulty with test items related to module 10 readily surpass the criterion 80-percent performance in items related to module 11 and to signed numbers. Here the test format and computer data processing enable identification of the difficult materials and may obviate the need for subsequent modules.

SIGNIFICANCE

Two statistical tests have been performed on M-31 data obtained to date. One-way analyses of variance have been done on each column (Figure 3) of group subtest scores to check for significant differences between groups, for example to see whether the respiratory therapy students and the LPN's could be regarded as coming from the same population. The analyses were run twice, once with and once without the health career preparation group. The results are presented in Table I in terms of F. values.

SUBTEST	WITHOUT HCP	WITH HCP
1 Whole Numbers	0.545	5.271*
2 Fractions	2.099	4.473*
3 Decimals	1.581	7.425*
4 Percents	5.246*	7.455*
5 Ratio	4.172*	7.483*
6 Signed Numbers	7.747*	8.135*

*Significant at 0.01 or less

Table I. F-values for analysis of variance in subtest scores, with and without health care preparation students.

These values show that, excluding the HCP students, only marginal differences appear between groups with regard to whole number, fraction, and decimal subtest scores. Significant differences appear in the other subtest scores. When the HCP students are included, significant differences appear in all subtest scores.

APPLICATION OF RESULTS

Using the procedures discussed, students are able to review their own strengths and weaknesses related to mathematics, both as individual applicants and in relation to the other applicants seeking admission into a program area. The Indiana Vocational Technical College at Indianapolis helps students in their areas of deficiency so they can develop the mathematical skills necessary for admission and eventual success. If scores are acceptable for program admission, the Related Education Department can use the data as a basis to increase mathematic skills in light of the health programs needs and objectives.

When imbalance develops between the numbers of students seeking to enter different health specialities, an individual's scores may prompt encouragement for consideration of other specialities also within the student's present or future range of mathematical development. For example, some students seeking to enter the practical nursing program may beneficially consider other instructional programs.

CONCLUSIONS

A computer-based testing procedure has been adapted to manage the mathematics placement of students pursuing instruction in health occupations. The application enables more efficient use of school personnel and expedites placement in, or by-pass of, a sequence of mathematics instruction modules. Some normative data have been obtained and some significant differences seem to exist between scores of students seeking instruction in different health specialities.

REFERENCES

1. Streeter, James and Alexander, Gerald. Fundamentals of Arithmetic. New York: Harper and Row Publishers, 1975.
2. Boyle, Thomas A. and Shaver, Carroll G. Computer Assessment of Mathematics Skills for Students Beginning Post-Secondary Technical Evaluation. Seventh Conference on Computers in the Undergraduate Curricula. Binghamton: State University of New York, 1976.
3. Boyle, Thomas A. and Wright, Gary L. "Computer-assisted Evaluation of Student Acheivement." Engineering Education, Vol. 68, No. 3, December 1977, pp. 241-245.

Invited Sessions

IMPROVING UTILIZATION OF TWO-YEAR COLLEGE COMPUTER CENTERS

Robert L. Burrows
Triton College
2000 Fifth Avenue
River Grove, Illinois 60171
(312) 456-0300

ABSTRACT

This discussion will begin with an overview of the areas of responsibility of a computer-assisted learning agency and then elaborate on what can be done in these areas once one actually becomes an agency. Specific topic areas include the following:

- --Marketing approaches to attract teachers to the computer via such means as seminars, newsletters, and committees.
- --Training instructors in computers, using as specific examples the three courses now being offered to Triton faculty each semester: Introduction to Computers and Programming, Introduction to Statistical Analysis Using SPSS, and Introduction to Computer Graphics.
- --Working with faculty to obtain educational software including catalogs, magazines, educational suppliers, and user groups.
- --Supporting software written or planned for academic users, including changes to system software, program filters written to convert foreign software, an electronic mailing system, a program directory system, and a CMI system.
- --Working with faculty in obtaining hardware, using as example the terminals and microcomputers purchased at Triton and its plan for a computer lab.
- --Problems with computer education specific to a community college.

TEACHING COMPUTER ETHICS

Walter Maner
Philosophy and Computer Science
Old Dominion University
Norfolk, VA 23508

ABSTRACT

The management, staff, and users of information systems of all kinds will benefit greatly from training in applied professional ethics. These professionals work in an environment where they must deal responsibly and knowledgeably with critical moral problems (such as breach of privacy, computer crime, and dehumanization) which are aggraveted, transformed, or created by the advance of computer technology.

A general rational for a course in computer ethics will, therefore, be developed along with course design criteria, a full course description, a set of proposed course objectives, associated bibliographies, and a taxonomy of subject matter areas within the field of information ethics.

Attendees will break into small discussion groups to study cases illustrating moral dilemmas posed by the use of computers in education.

Tutorial

PASCAL TUTORIAL

Harry P. Haiduk
Amarillo College
P.O. Box 447
Amarillo, Texas 79178

ABSTRACT

This tutorial is concerned with

(1) a brief historical view of PASCAL in terms of its philosphy and stated design goals

(2) a brief review of its current relevance in a diverse set of applications, particularly as it may relate to the new Department of Defense Common High Order Language, ADA

(3) a comparison of PASCAL's logic and data structures with those of BASIC, COBOL, FORTRAN, and PL/I

(4) actual running program examples contrasting PASCAL with BASIC and FORTRAN.

Pre-College Instructional Materials

COMPUTER-BASED INSTRUCTION FOR THE PUBLIC SCHOOLS: A SUITABLE TASK FOR MICROPROCESSORS?

Dr. Timothy D. Taylor
Computer Based Education Center
308 Carroll Hall
The University of Akron
Akron, Ohio 44325
(216) 375 7848

OVERVIEW

The microprocessor is proving itself as an attractive, low-cost device which has the capability to serve its owner as a record-keeper, accountant, entertainer, and tutor. The attractiveness of the device has led teachers and other educators to explore its potential for providing computer-based instruction. While no device, whether it be a microprocessor, a large computer, or something in between, should be viewed as simply good or bad for education, all devices have characteristics which must be considered before an investment is made. One might expect that a computer terminal connected to a large computer (resident or non-resident) might have capabilities that microcomputers lack, and this is certainly the case. Whether or not a school should purchase a micro, however, depends entirely upon the function which the machine will have. Following is a discussion of the educational services that can and cannot be provided by off-the-shelf, low-cost, microprocessors such as the Apple II or Radio Shack's TRS-80 (Model I).

DISPLAY CHARACTERISTICS

The variety of characters, graphics, and colors that microprocessors are capable of displaying has been a major factor in the success of its sales during recent years. The Apple II and Ohio Scientific micros, for example, can produce graphics, colors, and sounds, in addition to displaying standard keyboard characters. Graphics are often helpful, particularly for the teaching of such subjects as physics and higher level mathematics, but the most important characteristic required for much of the current instruction in the public schools is the display of upper- and lower-case alphabetic characters. None of the low-cost microprocessors is sold with upper- and lower-case displays, although most of them can be modified, at extra cost, to add this capacity. The use of color and sound can add significantly to the appeal of microprocessor-based courseware, but the lack of dual-case alphabetic characters has to be considered a serious limitation for the teaching of such subjects as English and reading.

SIMPLICITY OF OPERATION

A public school student using a terminal tied to a large computer typically begins his session by being seated and typing one short message. At that point, he has entered the computer-based education course and is working where he left off during his last session or where his previous performance records indicate that he should be. One CBE course may contain enough material to teach the student and track his progress for a year or more. A student using a microprocessor (presumably with at least one disk drive) typically has a more complicated procedure for beginning his session. He will obtain his disk, be seated, insert his disk into the disk drive, power-on the machine so that the initial program will be loaded into memory, and then he is ready to begin. He may or may not have to type a message telling the computer to start the lesson. He probably cannot start where he left off last time; however, a carefully programmed lesson may enable him to choose his own starting point. One disk may contain enough material to tutor the student for a number of hours.

In the public schools, students using terminals connected to a large host computer can be given the freedom to use the terminals without teacher supervision because the sign-on procedure is simple and because no peripheral materials, such as floppy disks, are required. Use of a microprocessor may require the monitoring of a disk storage area and a check-out, check-in system for students to borrow disks. Since a standard 5-inch floppy disk contains only a limited amount of material, the student will frequently need to exchange his disk in order to switch to another topic.

In terms of simplicity of operation, the advantage appears to rest with the terminal connected to a large host computer.

ADDRESSING A VARIETY OF LEARNING NEEDS

A typical, well-planned CBE course contains a sufficient quantity of instruction to meet the needs of a variety of different students: the slow learner, the rapid learner, the student who knows nothing about the topic being presented, the student who already has familiarity with the subject matter. As the student works through the instruction, his performance will cause him to be branched forward for advanced material, backward for additional practice, or laterally for a discussion of topics tangentially related to the primary topic being presented. The number of paths that a student could take through the material is infinite because the structure provides almost limitless branching opportunities. Since no two students possess identical needs, it is probable that no two students will progress along the same path of instruction. A computer-based course should be extremely flexible, broad in scope, and capable of handling students' individual performance styles. Failure to attain these characteristics constitutes a failure to place CBE into a category separate from other media such as lectures, textbooks, and audio-visual tapes, all of which possess a predominantly linear flow.

The above characteristics make present computer-based instruction impractical, if not impossible, on today's most popular microprocessors. Microprocessor tapes, disks, and memory have a size limitation which prohibits them from delivering a CBE course with the breadth of scope described above. At best, a student using a micro for this type of instruction would find himself swapping disks constantly in order to branch ahead, backward, and laterally. But microprocessor courseware with this degree of individualization is not available today. The educational software currently available for microprocessors consists of short lessons, each addressing a specific topic. Seldom does any continuity exist among these short lessons. To employ these microprocessor lessons for genuine computer-based education would mean that decisions as to which students need which lessons at what time would be made by teachers, aides, or by the students themselves. Whether these persons would make the correct decisions, and whether continuity could be provided among the lessons collected, is questionable. Assuming that continuity could be achieved, students would have to be given paper-and-pencil tests constantly in order to determine which lessons were needed by each student and in what sequence. Such tests are unnecessary on a large computer system since branching occurs automatically as a result of present and past student performance. In a well-designed CBE lesson, students are generally unaware that such intricate branching is taking place. They know only that the computer is moving in a logical direction from the beginning to the end of topics that they need to learn.

TRACKING STUDENT PROGRESS

Microprocessors and large CBE systems both have the capacity to provide on-line assessments, to give the student some information about his progress, and to branch him to various locations based on his performance. (As mentioned before, the physical size limitations of the microprocessor's disk would frequently necessitate disk-swapping in order to switch topics.) But in order to build a genuine performance history on each student, a large system is required. Large-system CBE can relieve the teacher of the burden of administering periodic tests and plotting each student's growth. By typing appropriate keywords at the computer terminal, the instructor can view immediately any student's current and past performance records. Typical information includes the following:

1) the student's first and last date on-line.
2) the total number of hours and minutes spent.
3) topics where mastery has or has not been demonstrated.
4) areas of particular difficulty.
5) areas not tutored due to excellent pretest performance.
6) average response time on drill-and-practice items.
7) length of individual sessions.
8) students' comments on parts or all

of the instruction.
This information can be displayed for one student, a group of students, or all students in a teacher's class. Hardcopy terminals are often used so that the report may be kept for future reference. Teachers can quickly identify any students having difficulty, and students can become very highly motivated when they see that they have achieved progress from one week (or month, or year) to the next.

The quantity of student performance data afforded by the microprocessor is trivial when compared to the data routinely kept on the large system. Many (perhaps most) of today's tutorial microprocessor lessons give no performance statistics whatsoever. Those lessons that do give performance information do not store it permanently for the teacher's later inspection. Performance data are collected and reported to the student as he is working, but when his session is over and he removes the disk from the disk drive, all collected information is instantaneously lost. Although virtually no microprocessor courseware exists that builds a history of student performance, it is possible to do so. The most practical way to accomplish such a task is to use a multiple-disk system. In a two-disk system, for example, one disk can be devoted to collection and storage of performance data while the other contains lessons being presented. In a public school, each student would be assigned one or more disk for storage of his own performance data, and these disks would be checked in and out along with the disks containing the instruction. Instruction would have to be programmed carefully so that each lesson would store performance data on a different part of the data-collecting disks. An instructor wishing to view the performance histories of his 30 students would sit at the microprocessor with his collection of 30 or more disks and slip them in, one at a time. As each student's progress is displayed, the information could be duplicated in hardcopy form if a printer is attached to the microprocessor. Obviously, this inspection of student progress would be a time-consuming task, and the performance data probably would not be as complete as the data collected on a large system.

It was mentioned above that typical performance data provided on large CBE systems includes the student's first and last dates on-line, total time spent on instruction, and average response time on drill-and-practice exercises. Unfortunately, many of today's popular microprocessors cannot record any of this information. It is possible to ask the student what the date is or how long it took him to answer a question, and then the student's response can be recorded, but the reliability of such data would have to be questioned. The absence of an internal clock precludes the recording of dates or elapsed time by the microprocessor. Other types of data, such as topics mastered and scores earned during on-line assessment, can be recorded by the microprocessor if the lessons are programmed to do so and if a system of data-collecting disks, like the one described above, is established.

In short, the collection and reporting of student performance data are essential in order to track progress, provide student motivation, and isolate learning problems. Large CBE systems address this need very well. Microprocessors could provide some (but not all) of the same student performance records only after a fairly complicated system of specially written lessons, student data disks, and multiple disk-drive hardware is created. Retrieval of information from this system would be significantly more cumbersome than it is presently on larger CBE systems.

INSTRUCTIONAL LANGUAGES

Like spoken languages, computer languages are numerous and varied. Many computer languages perform similar tasks equally well, but some languages are specialized to perform certain tasks better than the others.

Computer languages that have been designed to provide instruction are sometimes referred to as instructional languages. Such languages typically record some basic information automatically for every student on every CBE course, such as his first and last usage dates, his total time on each lesson, and his current location within each lesson. In addition, storage facilities are provided to enable the course author (programmer) to store an infinite variety of performance data permanently or temporarily. A student progress report is merely a systematic display of these storage facilities. An instructional language also permits the course author to accept a wide range of responses from the student. For example, if the author expects "false" as the answer to a question but realizes that students may misspell the word, he can use a statement which essentially says, "If the student types a five-letter word beginning with 'f' and ending with 'e', I recognize this as the correct answer." Another characteristic some instructional languages have is the automatic re-starting of the student at the point where he left off during his last session. Characteristics such as these are not found in computer languages that have been designed for purposes other than instruction.

BASIC is an example of a non-instructional language, not because it is deficient in some way, but because instruction is not the purpose for which it was created. It is a general-purpose computing language which is standard on virtually all microprocessors. The language has no pre-defined storage area for recording student performance data, no provisions for accepting a wide range of responses, and no automatic restarting of the student where he left off last time. Naturally, the BASIC language has capabilities that some instructional languages lack, notably in the area of mathematical computation. But if BASIC is to be used on microprocessors for CBE, numerous sophisticated subroutines must be created in order to give BASIC the characteristics of an instructional language. These subroutines will occupy space on the disk and permit less space for instruction and collection of performance data. The subroutines must be duplicated and stored on every disk used for instruction. And, in spite of their sophistication, the subroutines will not permit the recording of certain types of data such as dates and student response time unless the microprocessor chosen contains an internal clock.

As mentioned above, most languages can accomplish most tasks. But if a choice is made available, it is only logical to pick the language which is least cumbersome. On large systems, the choices are numerous and include many instructional languages. With microprocessors, the choices are few or nonexistent and do not include instructional languages. On the basis of languages, it appears that present-day microprocessors are not yet prepared to handle the level of sophistication found in large-system CBE courseware.

LARGE-SYSTEM AND SMALL-SYSTEM CHARACTERISTICS

Regardless of the language employed or the performance data collected, there are certain inherent characteristics of large systems which must be evaluated prior to making a decision to employ or abandon such a system. On the negative side, when the large computer malfunctions, all terminals connected to it become useless. But that all terminals are using one computer also has advantages. A major advantage is the ability to avoid duplication of effort. For example, when a courseware revision becomes necessary on a large system, one correction constitutes a system-wide change. But if 50 microprocessors were used in place of a 50-terminal network, then 50 individual corrections would need to be made--one for each copy of the defective course. Such duplication of effort is a general characteristic of microprocessors. Each user has his own computer and his own copy of each program he uses. Fifty users wishing to use a mathematics lesson would need to purchase 50 copies of that lesson. On a large system, only one copy of the lesson is needed, and it can be used on any number of terminals simultaneously.

Another important feature of large systems is that all users are accessing the same disk storage area. It is this characteristic which permits teachers to obtain class reports on groups of students Such reports often point out the best, worst, and average performance record of the group (sometimes including statistics such as mean, range, and standard deviation), thus giving the teacher an immediate picture of the group's homo or heterogeneity. With micro-based CBE, performance data are likely to be scattered across many disks, thus making such group reports impossible or impractical.

Large systems often possess a message-handling capacity which is not possible on microprocessors. Messages can be sent between course authors and students, for example. A student who is puzzled or who wishes to communicate with the author of his lesson for any reason can usually log a comment which the author will later read and answer. Such communication assists the author in locating any areas of his lesson which may cause confusion among students and enables teachers or other supervising persons to post administrative announcements to users of the system. Some large systems also permit on-line direct communication between terminals.

Microprocessors necessitate a duplication of effort and lack message capabilities, but they do provide a greater degree of independence and self-sufficiency. As mentioned above, a computer malfunction on a large system renders all terminals useless simultaneously, but one microprocessor's failure has no influence on other microprocessors. A micro is also more portable than a terminal connected to a large computer since the latter requires either a hardwired communications line or a telephone line. And, of course, there are no monthly rental charges after a microprocessor and peripheral equipment are purchased.

WHY BUY A MICRO?

A microprocessor can serve as a valuable educational tool, particularly in teaching students about computer hardware, and can help them develop an understanding of what happens when a program is written and executed. Mathematics students can also profit from a study of the BASIC language, which can be used effectively to solve math problems using algorithms. Limited amounts of instruction, of course, can be provided in tutorial or drill-and-practice form as well. But nothing with the scope, complexity, and record-keeping abilities of large-system computer-based education can be accomplished in a reasonable way on today's microprocessors.

LAUNCHING A CBE PROJECT WITH MICROPROCESSORS

Assuming that a public school has considered the pros and cons of the low-cost microprocessor as well as those of the CBE terminal connected to a large system, and assuming that the micro was chosen, these are some of the steps that would probably have to be taken:

1) purchase a microprocessor.
2) purchase at least two disk-drives in order to provide the capability for storing and retrieving student performance data.
3) purchase floppy disks for data storage and for storage of the instruction itself.
4) purchase a printer in order to provide student progress reports in hardcopy form.
5) modify the microprocessor so that it will display upper- and lower-case alphabetic characters needed for the teaching of spelling, English, reading, and related subjects.
6) design machine-language or BASIC subroutines to handle tasks already designed in instructional languages.
7) copy these subroutines onto all disks to be used for lesson content and/or all disks to be used for data collection.
8) initialize each disk to be used for lesson content or data collection.
9) build courseware which branches forward, backward, and laterally to accommodate individual learning styles and which keeps detailed student performance records on the data-collecting disks.
10) establish and supervise a system for storing lesson and data-collecting disks, keeping track of which disks are borrowed (and returned) by whom, and assuring that each student has access only to his own data disk.
11) make copies of lesson disks if the project involves more than one microprocessor.
12) when errors are detected, make the corrections on all disks containing the defective lesson or data-recording routine.

The above steps vary in complexity and would probably require years to accomplish. The characteristics of such a system are contrasted with the characteristics of present day large systems in the Summary Chart which appears at the end of this article.

NOTHING IS IMPOSSIBLE

It is not the author's intention to state that microprocessors are useless for computer-based education. In fact, most of the undesirable characteristics mentioned can be rectified through hardware modification, addition of peripheral devices, or by simply purchasing a larger microprocessor. But it is the small, comparatively unsophisticated microprocessor that has been attracting the most attention in the marketplace. This author becomes concerned when teachers and other consumers become entranced by those very attractive devices and when salesmen with little or no experience in computer-based education make exaggerated claims. Quality microprocessor courseware which has been tested objectively and shown to be of educational value is NOT widely available. Microprocessor courseware which provides the individualization and record-keeping of large systems is not available at all. Although it IS possible to buy a microprocessor with upper- and lower-case alphabetic characters, and although larger disk drives CAN be used so as to reduce the amount of disk-swapping, and although an internal clock CAN be added, such modifications result in a home brew system which is costly and which makes the in-house development of courseware mandatory. It is this message that the salesmen are failing to convey.

THE FUTURE

Microprocessors will probably play an important role in future CBE systems for at least two reasons:

1) Their capabilities will increase. It is hoped that a greater choice of computing languages will be available along with greater storage capabilities and, consequently, less required disk-swapping. An internal clock will enable the micro to record dates and keep track of elapsed time. Upper- and lower-case characters will be commonplace.

2) Downloading will become more practical. Downloading essentially means that the micro can communicate with a larger machine for the purpose of copying a program into its own memory or onto its own disk. Then the program can be used locally, independent of the large machine. After the session is completed, the updated

records can be transmitted back to the larger computer.

It is rather impractical to attempt to emulate large-system CBE with today's smaller microprocessors. A tremendous amount of time and effort would be required, and the final product would contain built-in disadvantages. By the time such a system is built, better micro-based systems will probably be available. At that time a re-evaluation of the enhancements made on microprocessors as well as on large systems will be necessary.

DESIRABLE CHARACTERISTICS

Typical Large System	Low-Cost Popular Micro	
	X	Involves a one-time cost without monthly rental charges.
	X	Displays graphics and colors, produces sounds.
X		Displays both upper- and lower-case alphabetic characters.
	X	Has complete portability, not dependent on a telephone line.
X		Requires one step (one short message) to start instruction.
	X	Affects only one user when the computer malfunctions.
X	X	Provides performance feedback while the student is on-line.
X		Provides detailed performance history for teacher's later inspection.
X		Enables students to continue where they left off during their last session.
X		Records dates and students' response times.
X		Records session time and permits instructor-defined session length for individuals or for all students.
X		Generates class reports showing histories of groups or students.
X		Has well-constructed courseware immediately available.
X		Is a system well-adapted for extensive branching to accommodate individual learning styles.
X		Supports instructional computer languages.
X		Permits messages among students, authors, teachers, and others.

UNDESIRABLE CHARACTERISTICS

Typical Large System	Low-Cost Popular Micro	
X		Involves a monthly rental charge.
X		Lacks graphics, colors, and sound.
X		Requires a dedicated line or telephone line.
	X	Has upper-case display only, unless modified.
	X	Requires executing several tasks in order to start instruction.
	X	Requires constant disk-swapping if individual learning styles are to be accommodated.
	X	Requires a disk storage and check-in facility which must be supervised.
	X	Necessitates adding more disks proportional to the quantity of performance data desired.
	X	Requires creating new courseware since virtually none exists presently.
	X	Necessitates considerable duplication of effort.

MICROCOMPUTER/VIDEODISC CAI
DEVELOPMENT CONSIDERATIONS

Ron Thorkildsen
Kim Allard
Exceptional Child Center
UMC 68
Utah State University
Logan, Utah
(801) 750-3534

Providing an appropriate education for all children in the least possible restrictive environment will require drastic changes in our educational processes, from assessment and prescription through instruction and monitoring. Recently Benjamin Bloom has stated, "If we are convinced that a good education is necessary for all who live in modern society, then we must search for the alterable variables that can make a difference in the learning of children and adults in or out of school" (Bloom, 1980). The alterable variables Bloom emphasizes are time-on-task, cognitive entry, and formative testing. These variables help explain the learning interactions between teacher and student. Using these variables implies individualizing the entire educational process, from assessment to the monitoring of learning.

Mainstreaming is currently viewed as a necessity in providing the least restrictive environment. Mainstreaming handicapped students creates a special burden for the classroom teacher, however, introducing handicapped students into the regular classroom greatly increases the range of intellectual capacity and experience of the students. This broadened range makes the individualization of instruction a necessity, although individualization will be extremely difficult considering the limited resources available to the regular classroom teacher.

Educators have long felt that the computer holds a special promise in providing individualized instruction, but this promise has not been fulfilled because of the limitations of audio-visual hardware and the high cost of computers. Recent changes in hardware should alleviate many of these limitations. The microcomputer has greatly reduced the cost of computing and the videodisc has the capacity to provide the audio and visual components necessary for effective individualized instruction via computer-assisted instruction (CAI).

A CAI system utilizing a microcomputer and videodisc is currently being developed as part of a research project conducted by Utah State University's Exceptional Child Center. The system is designed to be used by nonreaders and specifically by mentally retarded children and adults. Even though the system is designed to be used by the handicapped, it will have all the components necessary for a general purpose CAI system. With the appropriate courseware the system would be relevant to learners at any level of intellectual ability and experience.

The system, referred to as the MCVD (Micro-Computer/VideoDisc) System, was originally developed from funds of a small grant received from the University Research Office. Subsequently a grant from the Media and Captioned Films Division of the Bureau of Education for the Handicapped of DHEW was awarded. The grant is for two years and began October 1, 1979.

The major goal of the project is to develop, evaluate, and demonstrate a CAI system for use with mentally handicapped learners. The system is unique since it can communicate with nonreaders, a capability nonexistent or very limited in past CAI applications. Computer-assisted instruction systems have been devised and developed to communicate with nonreaders, but in most cases they have used serial devices such as audio tape recorders and slide projectors. The difficulty with these systems is a relatively long access time when branching to different segments of an instructional program. A short access time (less than 3 seconds) is critical in most CAI applications, but especially critical when working with mentally handicapped learners since, at best, their attention is difficult to maintain.

The hardware for the MCVD system consists of the MCA videodisc player, an APPLE II microcomputer with a digital disk, a Sony 12-inch television monitor, and a Carroll Manufacturing light-interrupt touch panel. The MCA 7802 videodisc player was selected because of its random access capabilities, and the APPLE II microcomputer system was selected because of its portability, reliability, and color graphics capabilities. A color monitor was chosen because of the additional flexibility provided by color. The touch screen is a light-interrupt system which allows the learner to interact with the system by touching the screen. The hardware components are enclosed in a cabinet that disassembles for transportation purposes. The monitor will be mounted on an adjustable pedestal that swivels, tilts, and can be adjusted for elevation. This flexibility and adjustment is necessary to accommodate the wide range of ages and varying degrees of motor ability. The cabinet is mounted on retractable wheels to facilitate mobility.

The computer programs that control the videodisc and receive input from the touch panel are written in Applesoft BASIC. These programs were originally written in PILOT, which we abandoned because of problems with the PILOT translator. PILOT was originally selected because of its ease of use. We are currently exploring various PILOT translators and compilers. We have determined that a PILOT interpreter does not provide for sufficient execution speed to interact with the three devices. We are also currently investigating the potential of PASCAL as a language for the system.

Communication between the three hardware devices is accomplished through an interface board that was designed and built by USU's microcomputer lab. The interface board also can programmatically switch the video source to the monitor from either the APPLE II or the videodisc. This capability allows us to transmit APPLE II graphics and audio to the monitor while the videodisc is searching. It also allows us to present video information from the APPLE II and audio from the videodisc simultaneously, which provides for a great deal of flexibility in altering the video images from the videodisc.

At present the system interacts with the learner by presenting an audio instruction and the associated visual image on the monitor via the videodisc. The learner responds by pointing to an object on the monitor screen. When the learner touches the screen, two light beams transmitted from each axis of the touch panel are interrupted, and the point of interruption is detected by the touch panel. The X and Y coordinates from the point of interrupt are transmitted to the microcomputer. The computer program in the microcomputer contains the correct coordinates for each segment of instruction, and the coordinates transmitted by the touch panel are compared to these correct coordinates. If the response is correct, the microcomputer responds by referencing the segment on the videodisc which contains audio and visual positive feedback. Other possible response conditions are a wrong response, a close response, and a non-response. (A non-response is detected when the learner does not respond in a specified period of time. Recorded segments are contained on the videodisc for these response conditions as well as a variety of feedback, including animation and motion picture sequences. Each segment of instruction has associated parameters that specify the number of times a learner must respond correctly to advance to the next segment and the number of trials allowed before the teacher is signaled for help. While the learner interacts with the system, data are collected by the microcomputer and stored on the APPLE II magnetic disk.

Various data are maintained which track a learner's progress through a program of instruction. The data are then available for analytic purposes and are also used by the system to allow the learner to start at a point where the previous session was terminated. This automatic restart option can be overridden by the teacher, who can specify the point at which the learner is to continue. The data is readily available in hardcopy form via a Centronics Printer.

Four instructional programs are currently in various stages of development. These instructional programs are developed by the Outreach and Development Division of the Exceptional Child Center and have been field tested and validated. They were chosen primarily because the instructional sequences had been validated and because they are adaptable to the MCVD format. The following is a brief description of the programs:

1. Matching Program (Hofmeister, 77) - This program teaches the child to match objects that are alike in size, color, or shape such as identifying squares with other squares. This skill is necessary before the child can learn the more advanced skills such as names of colors and shapes and reading. This program was not designed to teach a child to to name colors, sizes, or shapes.
2. Timetelling Program (Hofmeister, 75) - This program is to be used with any child or adult who cannot look at a clock and recognize what time it is to the minute. It has been especially useful in tutoring programs for the slow learner, although it is not restricted to such use. Upon completion of this program, the learner should be able to look at any clock, showing any time, and recognize the correct time to the minute.
3. Recognition of Functional Words (Hofmeister, 76) - This program provides instructions for teaching the learner to recognize functional words. This program teaches the recognition of twenty different words important to everyday living (e.g. stop, go, pull, push, etc.).
4. Identification of Coins (Hofmeister, 77) This program teaches the learner to recognize coins upon sight.

Preliminary development of the Matching Program was completed in May 1979. A preliminary field test was conducted and changes were made to the system as a result of this field test. A second preliminary field test was conducted and changes were made to the system as a result of this field test. A second preliminary field test of the Matching Program is currently being conducted. It was anticipated that this field test would be completed by the submission time of this paper, however, the field test will be continued through February 8, 1980, in order to collect additional data. This second field test was necessary to: (1) evaluate the changes in the system that were precipitated by the first field test; and (2) further define the population.

The following problem areas were determined as a result of the first field test:

1. Even though the search times were relatively short (less than 2.5 seconds), the attention of the child was lost when the monitor went blank during a search time.
2. The system was giving negative feedback to a correct response when the child was required to touch multiple objects on the screen.
3. The child seemed to lose interest in specific positive feedback segments if they were repeated several times.

In an attempt to rectify these problems a number of changes were made to the system:

1. A programmable switch was added to the interface board that allowed the computer program to switch the source of video from either the microcomputer or the videodisc. This switch enabled the system to present a computer-generated graphic while the videodisc was searching.
2. The algorithm in the computer program used to detect multiple responses on the screen was changed. We determined that the computer program was not responding rapidly enough to detect rapid responses from the learner. The algorithm was changed, and a rapid increase in response time was realized.
3. The algorithm used to select positive feedback segments was changed so that the segments could be presented randomly.

The six children involved in the second field test are from the special education classrooms at the Exceptional Child Center. These children are functioning at a lower mental development level than those involved in the first field test. The purpose of this change was to determine if children at the lower level of development could effectively interact with the system. The only criteria established for selection was that the children have sufficient motor skills to hold and point with a small pointer and sufficient receptive language to understand simple commands such as "touch the one like this," "that was good," "that was not correct," and "try again."

The field test has been in operation for three weeks at the time of writing. During this time numerous problems in working with the test population have been identified. Since the field test is continuing, the data cannot at this point be analyzed and specific conclusions cannot be made.

The first half of the Timetelling Program is planned for production in March 1980. The following entry-level skills have been established for learners using timetelling courseware (these entry level skills will define the population):

1. Learners must be able to count to sixty and recognize numbers 1-12.
2. Learners must be able to identify basic geometric shapes (e.g. circles, squares, etc.).
3. Learners must have sufficient motor skills to hold and use the pointer.

At the completion of the first half of the program, learners will be able to:

1. Sequentially count and place the numbers 1-12 around the clockface.
2. Recognize the digital format (e.g. 2:) for the little hand.
3. Determine the value of the little hand with digital cues on the clock face and the big hand as a distractor.

As a result of information gained during the second field test of the Matching Program, it was determined that physical prompting and practice in using the pointer are required before interacting with the abstract objects on the television screen. The Timetelling Program is being developed to involve the teacher (or another student) in the beginning phase of the program. When appropriate interaction with the CAI system is achieved, the system takes over, allowing the learner to work independently with the CAI system.

Based on the unique capabilities of the MicroComputer/VideoDisc System, the following courseware design considerations were formulated and incorporated into the development of the Timetelling Program:

1. Although the availability of a second audio track affords many instructional options, such as instruction with different voices or in a second language, as courseware development proceeded both positive and negative, it was determined the second audio track should be used to store specific feedback, to the learner on his performance (see Figure 1). This feedback is specific to a particular segment of instruction as opposed to the standard feedback blocks which are used by numerous instructional segments. Generally, it is the instructional sequences at the beginning of the program that require specific feedback. An example would be "good touching the one."
2. Strategic placement of the standard feedback blocks can substantially reduce the access time required by the videodisc player to search for a feedback segment. Because of the nonlinear nature of the MCVD system, standard feedback is used often throughout the program. A regression equation developed by Woolley and DeBloois allows the prediction of access time based on the number of frames (Woolley, 1980). A criterion of 2.5 seconds was established for the maximum search time for providing either positive or negative feedback. Based on the regression equation, the feedback segments would have to be located within 6900 frames of any instructional segments that access it.

Meeting our 2.5 second search criteria required multiple placement of the standard feedback blocks.

3. Periodic testing is facilitated by the nonlinear capabilities of the MCVD system. The timetelling courseware and microcomputer software are designed to branch to appropriate practice frames for criterion testing. This branching insures that the learner will be tested only on the material the learner has already covered. Criteria have been established at 80% accuracy. Testing, consisting of 8 frames, begins and continues only as long as the learner meets the 80% criterion. In other words, as soon as two frames have inaccurate responses, the software branches the learner back to the initial lesson for continued practice. Conversely, should the learner meet the 80% criterion on his or her seventh frame, the last test frame will be omitted.

4. The amount of repetition can be varied for individual learners based upon their previous performance. The learner who achieves a predetermined level of accuracy (determined during field testing) may be branched ahead to the criterion test and given the opportunity to demonstrate mastery of that particular concept. Learners having difficulty with the same concepts can be given additional repetition and subsequent practice.

The two preliminary field tests and production experience have yielded valuable information on the development of videodisc-based CAI. Presently one of the most limiting disadvantages of CAI involving the videodisc is the high cost of disc pressing. This presents instructional courseware developers with an entirely different set of instructional development problems. After a disc is pressed, you have to live with it. Therefore, any experience gained from this project or other projects should provide valuable information in developing videodisc programs. Essentially, a developer needs all of the up-front knowledge available before a disc is pressed.

REFERENCES

Bloom, B.S. "The New Direction in Educational Research." Phi Delta Kappan. 1980, Vol. 16, No. 6.

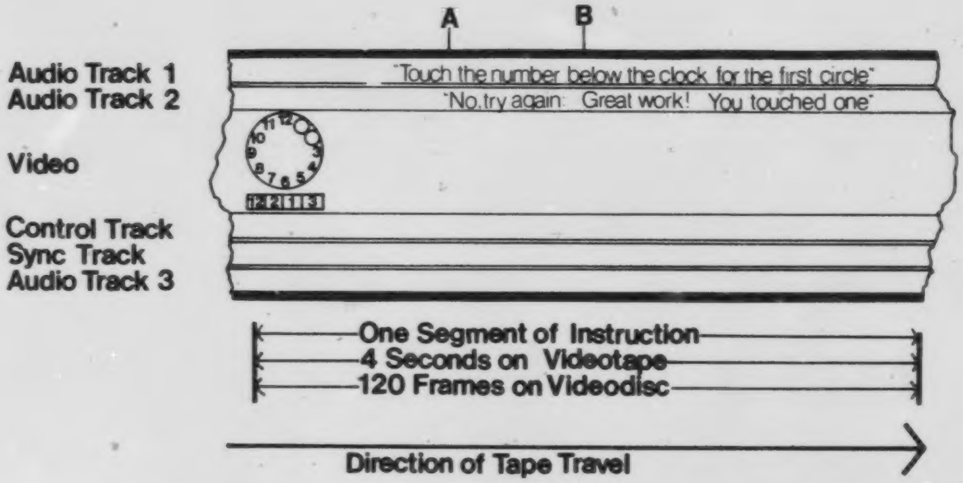
Hofmeister, A.M. & Gallery, M. A Program for Teaching the Identification of Coins. Niles, Illinois: Developmental Learning Materials, 1977.

Hofmeister, A.M., Gallery, M. & Landeen, J.J. "Matching Sizes, Shapes and Colors." An Instructional program prepared by the Exceptional Child Center, Utah State University, Logan, Utah for the Utah Division of Family Services, 1977.

Hofmeister, A.M., Atkinson, C.M. & Hofmeister, J.B. Programmed Time Telling. Eugene Oregon: E.B. Press, 1975.

Hofmeister, A.M., Patten, M. & Rosen, A. "A Parent Teaching Package - Word Recognition." Available at the Outreach & Development Division, Exceptional Child Center, Utah State University, Logan, Utah, 1976.

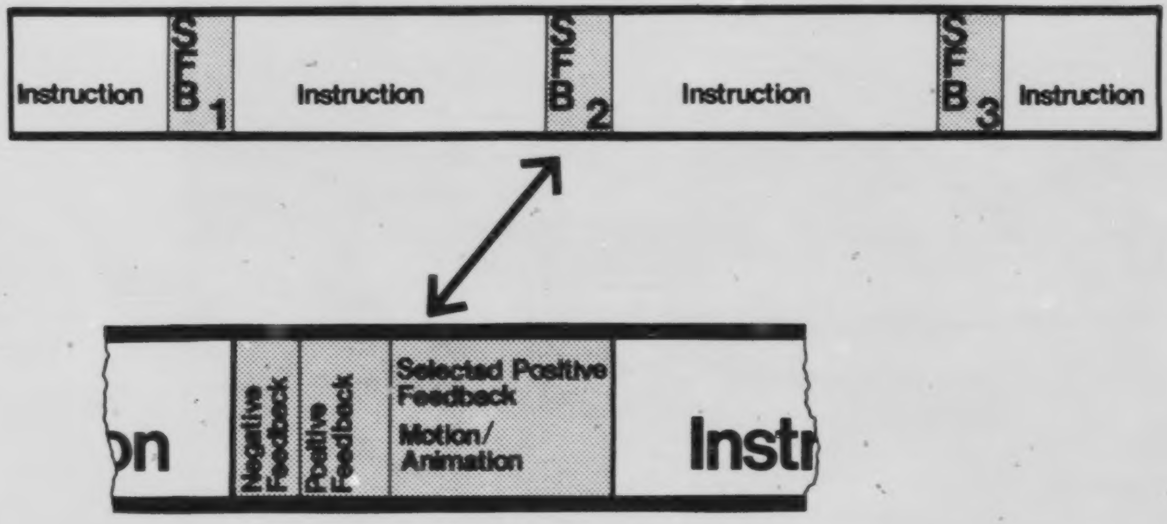
Woolley, R.D. and DeBloois, M.L. "Preliminary Benchmark Data for the PR #7820 Discovision Associates Videodisc Player." Center for Instructional Product Development Technical Report #1. Utah State University, Logan, Utah, 1980.


A
B
Audio Track 1
Audio Track 2
"Touch the number below the clock for the first circle"
"No, try again: Great work! You touched one"
Video
12 2 1 3
Control Track
Sync Track
Audio Track 3
One Segment of Instruction
4 Seconds on Videotape
120 Frames on Videodisc
Direction of Tape Travel

**One Inch Type C Helical Videotape
Used For Initial Production**

Figure 1

Linear Framework of Timetelling Package


Instruction
SFB 1
Instruction
SFB 2
Instruction
SFB 3
Instruction
on
Negative Feedback
Positive Feedback
Selected Positive Feedback
Motion/ Animation
Instr

Standard Feedback Block (SFB)

Figure 2

THE NON-TECHNICAL FACTORS IN THE DEVELOPMENT OF CAI

Michael Mocciola
Computing Center/Academic Computing
Pace University
New York, New York 10038

ABSTRACT

The most challenging opportunities and the most serious problems that inhibit the development of computer-aided learning in education are non-technical. In most educational institutions decision making, communications, and bargaining about the funding of computing, the purchase of computer equipment, or the application of computer methods is divided among the major populations of the schools: school board members, administrators, teachers, department heads, and students. To understand the socio-political factors related to computer-aided learning, the job-related responsibilities of each of these populations must be carefully examined. Computer educators then need to use the socio-political forces to the best advantage of students and teachers to provide enhanced classroom instruction. Educators must recognize and foresee the needs of the school community, have early access to pertinent information, and communicate effectively.

Invited Sessions

DATA SETS AVAILABLE FROM THE FEDERAL GOVERNMENT

Chaired by Thomas E. Brown
General Services Administration
National Archives and Records Service
Washington, DC 20408
(202) 724-1080

ABSTRACT

Many agencies of the U. S. Government routinely make computerized data available to educators and researchers. To do this, some agencies provide reference service for the data which the agency created, other agencies distribute data originally created by other agencies, and still other agencies serve as a clearinghouse for a given subject and refer researchers to agencies having information on that subject. This session will have representatives from three agencies performing these three functions. At the end of the session, one should have a clear indication of the procedures involved in locating and obtaining data sets from the Federal government.

SPEAKERS:

Les Solomon, Coordinator, College Curriculum Support Project, Bureau of the Census: a discussion of the services which the Bureau of the Census provides to enable researchers to gain access to data collected by the Census Bureau.

Ross J. Cameron, Archivist, National Archives and Records Service: an outline of the functions of the Machine-Readable Archives Division in its mission to inventory, obtain, and provide reference service for data created by other Federal agencies.

Edward D. Mooney, Program Specialist, National Center for Educational Statistics: a presentation on the Federal Interagency Consortium of Users of Educational Statistics whose purpose is to facilitate access to education data in the Federal government.

Minority Institutions—ECMI

ACADEMIC COMPUTING:
A SAMPLER OF APPROACHES
IN MINORITY INSTITUTIONS

Sister Patricia Marshall
Xavier University of Louisiana
7325 Palmetto Street
New Orleans, LA 70125
504/486-7411

A variety of post-secondary minority institutions using a variety of approaches to academic computing were interviewed campus-wide and in-depth late in Fall 1979. These interviews were part of a larger assessment of needs in educational computing at 239 minority institutions.(1)

Faculty, students, and administrators were interviewed on computing development, usage, problems, and successes. Diverse approaches were discovered, corresponding to philosophic, demographic, geographic, historical, political, and cultural factors.

The institutions were selected in order to obtain as broad a cross section as possible within time and financial limits and parameters such as ethnic composition, type of control, date of establishment, highest level of offering, academic orientation, enrollment, type of access to hardware, and experience. While not all of the most successful institutions were chosen, we did try to include schools which would not feel threatened by the interviews and which had had enough experience to identify factors inhibiting and promoting progress. Consciously excluded were such institutions as the Universities of Hawaii and Puerto Rico, partly for financial reasons and partly because they are so much larger than minority institutions in general. The table on the next page lists the institutions interviewed and some key parameters used in their selection. For confidentiality we identified them only by number. We will report here on four of them; these are named below with permission.

Institution #1

The Community College of Baltimore is an Eastern seaboard, public, urban, two-year, three-fourths black institution established after World War II. It enrolls over 9,000 students in day and evening divisions. Two campuses, one emphasizing technical studies and one evolving toward a focus on business studies, are united under one administrative structure. Large numbers of minority students began to attend this school early in the '70s as a result of outreach by the institution. The stable faculty consists of only 28 percent minorities. Although this institution may appear small by national standards, it is one of the largest of the minority schools and one of the few two-year colleges with any large degree of experience and planning in academic computing.

Unencumbered by the weight of tradition and spurred by local employment needs, the Community College of Baltimore had established a separate academic department for data processing by the early date of 1965. This department, now called Computer and Information Systems (CIS), attracts 300 majors and graduates about 25 each year with an associate degree in computer studies. Four full-time and numerous part-time faculty staff the department.

The school has weathered three hardware phases and is entering its fourth. Initially an IBM 1620, acquired for the data processing department in 1965, was the only computer. A committee from electronics, data processing, and administration hired a manager of computer services and developed specifications for a UNIVAC 9300, which arrived in 1968. The college allocated its funds to lease it and pay support staff for administrative services. Departments and individual users, however, have never been charged.

1972 brought a UNIVAC 9480 (131K) with

TABLE 1

MINORITY POST-SECONDARY INSTITUTIONS INTERVIEWED ON ACADEMIC COMPUTING, FALL 1979

INSTITUTION	DATE ESTAB.	ETHNIC TYPE			TYPE OF CONTROL		HIGHEST LEVEL OF OFFERING			ORIENTATION				FALL 1978 ENROLLMENT	TYPE OF HARDWARE ACCESSIBLE BY STUDENTS AND FACULTY
		B	S	I	PUB	PRI	2	4	4M	LIB ART	TECH /VOC	ENGINEER	CAREER		
1	1947	x			x		x						x	9,152	UNIVAC 9480 IBM 1620 APPLE II (5, used as terminals and stand-alones) DEC 10 (remote)
2	1873	x				x		x		x				600	HP 2000 IBM 1130
3	1969		x		x		x				x			876	DEC 10 (remote) PDP 11/70 (remote) PDP 11/34 IBM 360 (remote) IMSAI 8080 TRS-80
4	1891	x			x				x	x		x		5,395	DEC 10 HP 1000 Micros
5	1884			x	BIA		x						x	1,013	DEC 11/34 IBM 1401
6	1867	x				x		x		x				1,526	DEC PDP 11/34 IBM 1130
7	1968			x		TRIBAL	x						x	839	DEC PDP 11/45
8	1881	x				x			x	x		x		3,296	HP 2000 IBM 370/158 (remote) SOL (3, micros) SWTP (micro)
9	1966	x	x		x			x						4,315	IBM 3031 (remote) IBM 3033 (remote) AMDAHL 470 (remote)

NOTE: Campus-wide interviews were conducted at the institutions listed above as part of a needs assessment of educational computing in minority post-secondary institutions.

ETHNIC TYPES:
B - Black
S - Spanish speaking
I - American Indian

HIGHEST LEVEL OF OFFERING:
2 - Two years
4 - Four years
4M - Master degree

four terminals for administrative data entry and inquiry. COBOL, RPG, and FORTRAN were supported for students, as well as ASSEMBLER (on the 9480 and the 1620). This UNIVAC is still used for student batch runs. In addition, ten ports were rented in 1975 on a companion community college's HP 2000, an arrangement that was terminated in 1979 in favor of access to a DEC 10 at a nearby private university. Students use ten DecWriter terminals which are dedicated during labs for a course in BASIC and used for other courses during open times.

The CIS faculty purchased five microcomputers in Fall 1979 through a Title VI grant. These have replaced the 1620 in teaching assembler languages and computing principles. A fourth system is under consideration now because of the two campuses, growth of institutional and faculty research, and growing concern for academic computing. The institution had done its own local needs assessment before considering such a step.

Thus, the Community College of Baltimore has steadily built its computing capabilities, if at a slower pace than some majority institutions certainly also at a faster pace than most minority and/or two-year institutions. This stable growth has resulted from support by the college of increases in computing capabilities through its own funds, supplemented by aggressively sought federal grants. Because of its careful planning and budgetary practices, the school will be able to assume ancillary costs when grants expire.

Factors influencing the expansion of computing capabilities at this institution have been the local employment market, size of the student body (necessitating automation in record handling and giving early exposure to administrators and faculty), support by administration, key faculty members with interest and dedication to supplement the budget for computing resources, and explicit planning mechanisms within the college charged with studying computing needs.

Administrative support is evidenced by hiring practices, release-time practices (for proposal writing and planning), and activities involved in acquiring computing equipment. A few key faculty members have visited other successful sites, attended conferences (ECMI(2) and others) and conventions, served on committees in professional associations, sought external funds, and lent one another intellectual and moral support. The evaluator who conducted the interviews at this community college commented:

> These individuals represent a scarce resource at any institution and seem to represent a necessary if not sufficient condition for progress in academic computing. (A broad base of faculty awareness is probably not a necessary condition for growth in an institution's computing capabilities.)

Periodic ad hoc committees on computer utilization further exemplify the planning mechanisms for computing established within the college. Plans also exist to establish a line position for a director of information systems reporting to the president through a dean of planning, development, and communications. This new position would assume responsibility for academic, as well as administrative computing.

Tensions have existed between administrative and academic computing, as at many other institutions. The manager of computer services reports to the vice-president for administrative services. The CIS Department, on the other hand, is within the Division of Business, Secretarial, and Computer Sciences under the vice-president for academic and student affairs. Support staff have grown through several stages to the current manager of computer services, three programmers, two keypunchers, and two computer operators. No students are employed since the manager claims "it doesn't work."

Students submit batch jobs through a slot in a door. Output pickups are scheduled twice daily for fifteen minutes each. However, during the two lab hours assigned each course in a batch-mode language, turnaround time is closer to immediate. Students, who probably know of no other alternatives, accept the pickup arrangement, complaining only about the number of keypunches. Terminals are available for learning BASIC, and the APPLE microcomputers were made available in Spring 1980 for learning ASSEMBLER. The commercial option in CIS will be offered at the campus nearest the business community and the scientific option at the other campus. Academic applications exist also in the business department (managing data on patient care and student performance). CMI(3) is expected to be developed in remedial reading and CAI(4) in science courses after the new configuration is installed. Faculty with a good track record in attracting federal grants will be responsible for developing CAI at the science learning center. The project director for the science learning center had attended an ECMI conference and attributes his current grant, in part, to that experience.

Institution #2

Bennett College is a private, women's four-year, historically black, liberal arts college with a tiny, though stable, enrollment of about 600. Located in Greensboro, a mid-Atlantic city, the institution houses its computing hardware in one attractive location on campus. This hardware includes an 8K IBM 1130 batch system used for administrative purposes and student programming courses and an interactive HP 2000 with fifteen terminals used extensively for CAI tutorial and drill and practice in mathematics, English, and biology. Two of four keypunches are available to students.

Computer center hours are 8 a.m. to 10 p.m. (Saturdays on request), and consultation is available during these times. Freshmen receive computer ID numbers at the beginning of the school year, but upperclass students have open-shop, direct access to both computers. They run their own decks and retrieve their own output immediately; however, lines do develop when the business office is running at the same time on the 1130. Tables are provided students in the computer center for such use as examining output and correcting programs.

Reports on the success of the CAI project, in which faculty prepare courseware (or modify existing courseware) using IDF, have been given at CCUC(5) and ECMI(4) conferences; a great deal of consulting and sharing with other colleges has also taken place. In addition, this college has been cited as an academic computing exemplar by HumRRO(6).

Computing began at Bennett College eleven years ago. A rented teletype connected the college to TUCC(7) and provided for some instructional work, as well as for improving computer literacy of the faculty. Influential in establishing the initial capability were the president of the college (who later attended an ECMI conference) and the mathematics department chairperson (who later became the computer center director). A year later, the college purchased the 1130 through a federal grant in connection with another university. This computer was used, from the beginning, for both administrative and academic purposes. It was also used cooperatively by two other small, church-related, primarily white institutions in the same city. All three schools used it for registration processing. Five years ago a Title III grant made possible the purchase of the HP 2000 with fifteen terminals for CAI development. Basic skills disciplines accounted for early CAI use; CAI is built into course requirements in these disciplines.

The success here with CAI has given rise to new problems. Students need twice as many terminals, and the 1130 is too small and is no longer supported by IBM. Additional personnel trained in computing are needed. Release time for faculty is needed to develop additional CAI courseware. Despite the success, the support of the president, and the small size of the institution, some faculty are still unaware of the potential in their disciplines. In such a small school, this is seen as a problem by faculty who do use the computer.

Attendance at ECMI conferences helped to develop a strong team of faculty, however, who have used instructional computing heavily. Through a federal grant the institution began to share its expertise in Spring 1980 by conducting a regional conference similar to the ECMI conferences. Bennett collaborated with North Carolina A & T University (Institution #4) in conducting that conference. Thirty small colleges participated.

One faculty member cited CAI as extremely helpful to entering freshmen, so many of whom are in need of remedial work due to inadequate preparation in secondary schools. Students said they appreciated the CAI but not the downtime. The president of the institution would like to see separation of administrative and academic computing. Plans for upgrading the existing hardware are on the drawing board.

Despite its small size and its dependence on hardware and software that are less than state-of-the-art by today's rapidly changing standards, Bennett College has taken a position of leadership in the development of transportable courseware for use in basic skills courses. Success has brought with it new problems, but it has also nurtured confidence, plans, and determination to solve the problems.

Institution #3

The Rio Grande Campus of Texas State Technical Institute is a technical-vocational, two-year, Hispanic institution located in Harlingen, close to the Mexican border in the southernmost tip of the state. One of four such institutions comprising a system in Texas, this school was established on a World War II air force base in 1967. Its fast-growing student body numbered well over 1,100 at the time of the interviews, and some of the growth is in the data processing program.

Curriculum design is a high-priority and on-going activity at the school, as evidenced by its special staffing and the existence of school-industry cooperative committees and advisory committees. Annual evaluation ensures that curricula are up to date and graduates are well prepared for employment. (Some students command entry-level salaries as high as $18,500 without graduating.) Just minutes away from this institution are vast farmlands on which thousands of Mexican Americans barely survive. Thus the existence of an industrial corridor and this institution to serve it is pivotal to economic change in the area. (On the first day of the interviews here the Wall Street Journal stated that this region was one of the four fastest-growing industrial areas in the country.)

A new building houses the computing and electronics programs and computing facilities, as well as some other mathematics and science or technical programs. Nearly one-third of its 17,000 square feet is occupied by computing facilities and classrooms used in the industrial data processing program (IDP). Prior to the interviews (just two weeks after the move to the new building) less than a fourth as much space had been available.

IDP students dominate the facility, which, like the entire campus, is outstandingly clean and well organized. Terminals are dedicated to students, or to entire classes, from 8 a.m. to 5 p.m. Often the facility is open till 6 because the IDP chairman stays late. Since most students are Mexican-American undergraduates who live with their families in the area and are not accustomed to being away in the evenings, the facility is not kept open at night. A demand does exist, however, from working adults in the area for an evening program. Finances and staffing are the obstacles to be overcome.

Two instructors are employed in the IDP program, one the chairman. An additional slot is open but not filled. The two instructors bear heavy teaching and lab loads, with most emphasis on lab work. They share the burden of supervising the facility. Second-year students are trained not only to program (and maintain programs) for some administrative and academic applications, but also to assist as consultants to first-year students when instructors are unavailable.

The IDP chairman has spent much of his own time on outreach to other departments, providing demonstration projects, seminars, and classes. The degree of interest from other departments likely to use the computer for instructional purposes ranges all the way from "take it away" to "you can't begin to meet MY needs." Those least interested are traditional instructors. In programs using individualized instruction (or in which instructors have had previous experience), faculty are eager to attempt computer-managed instruction or computer-assisted test generation. Attitudes appear to stem from educational philosophies rather than from familiarity with areas of expertise, such as nuclear technology or mathematics.

Two weeks after the move to the new building, when the interviews were conducted, a terminal room contained ten CRT's, two teletypes, and a line printer in fairly constant use. A Radio Shack TRS-80 and an IMSAI 8080 microcomputer were also available. Six additional CRT's, a printing terminal, a digital plotter, and a tape drive were also being readied for use. Three keypunches with acoustical shells were available. Bulletin board displays included charts, lists, schedules, computer-generated Mona Lisas, and a sign proclaiming "The Dirty Dozen." ("The Dirty Dozen," it turned out, were the second-year students who had survived out of a much larger original field of beginners in the IDP program.)

The terminals were on-line to a DEC PDP 11/70 located in a department store 45 miles away. COBOL is not the latest (1969), and RPG card decks are sent to a DEC 10 at a university 50 miles away, with a turnaround time of weeks. But students are obviously learning and being hired. The IDP department, having grown from four students in 1974 to more than seventy at various levels of advancement at the time of the interview, has gone through several upgrades of remote connections. Current plans call for an on-site computer with 32 ports for student use. However, instructors who want to support individualized instruction, record keeping, and test generation in open-entry/open-exit courses feel they may still not have sufficient capacity since they would be in contention with IDP for use of the resources. Meanwhile, administrators are working on additional capacity for administrative work (remote access to a large state institution's network).

One instructor has done some work of his own, some of which was destroyed in one of the upgrades. Most recently he has been using a TRS-80 microcomputer on his own time. He envisions a cluster of micros in a classroom, which he sees as a cost-effective solution to his problems in a tradition-oriented department. The electronics program, which trains almost a hundred students for customer engineering on DEC equipment dedicated to the program, can use twice as much hardware. The chairman of this program, a former ECMI(2) participant, keeps up with hardware periodicals and literature but is far too busy to move into academic computing generally.

Students who are beginners tend to see no problems with existing hardware and software, but the advanced "Dirty Dozen" talk like data processing managers. Completely at ease in the jargon and what's behind it, they speak knowledgeably about the shortcomings of the available software, the need for this version of a language and that many ports, and even the additional justification needed in the current proposal for new equipment.

Administrators give moral support, but their budget requests have to move through several layers of state bureaucracy and compete with other technical institutions. Even when funds exist, it is difficult to find qualified staff in the area who are not already working for burgeoning industries at high salaries. Nevertheless, the industrial data processing program grows, and other departments are beginning to voice their needs. Among two-year technical institutions, TSTI-Rio Grande may be on its way to becoming an academic computing exemplar.

Institution #4

North Carolina A & T University is a state-controlled, four-year, master degree-granting, black institution, with liberal arts and some engineering emphasis. The enrollment is 5,400 students. Located across town from Bennett College (Institution #2), this school was the last (and only minority) institution to procure a mainframe computer through the National Science Foundation's original Office of Computing Activities. That computer was a CDC 3300, and it followed the first computer acquired in 1964, an IBM 1620. It was replaced by a DEC 10 in 1977. Federal and state funding was combined in each of the latter two cases, with the CDC being sold to add to the funding for the DEC.

Although some impetus came from the computer center (especially recently), a key role was played by former dean of arts and sciences Arthur Jackson, after whom the computer center is now named. More than 80 terminals are on-line to the DEC 10, 26 of them in the computer center. A variety of languages are available: BASIC, PASCAL, APL, COBOL, ALGOL, LISP, and others. The computer is available around the clock all week. The staff includes nine programmer/analysts, and experts are available to faculty and students as needed. Students access the computer interactively through class accounts or individually (both authorized by departments) as well as by batch jobs. A monthly report provides usage information but is not currently used for charging. Most use is by students in coursework, but faculty do research, and administrative applications abound. Turnaround time is good except at peak times. Jobs are limited to fifty at a time, which can cause a connection delay of ten-to-fifteen minutes. Work space in the computer center for students is also limited.

During the past two years North Carolina A & T University has experienced what the computer center director refers to as "a quiet revolution" in computing. From a single-job, batch-mode machine with 49K main memory and 25 megabytes of disk, the university took a quantum leap to 256K, 600 megabytes of high-speed disk and the ability to process up to fifty jobs at a time in many languages interactively, in batch mode, or in combination.

However, increased demand for computer time has lowered response from excellent to poor. Engineering, which six months before the interviews could run circuit analysis, finite element, operations research, and other sophisticated jobs at any time, must now normally execute many of these between 5 p.m. and midnight, and several between midnight and 8 a.m. Engineering school jobs also include CAI packages needed for educationally disadvantaged students, and it is difficult to find time during the day to run them. Plans are afoot for a memory upgrade at $100,000, half of which must be raised from outside the university budget. Further upgrades will probably include a remote-site laboratory and a processor upgrade. Staff and space needs were cited in addition to hardware and stations for remote access.

An active computing advisory committee, chaired by a chemistry professor, functions as an agent of change and supplements positive pressures from all types of users and administrators. Proposals have been written, and a computer science degree program is to be launched in Fall 1980. An academic computing director and an administrative computing director will also be hired.

Some usage came about through participation of faculty at ECMI conferences(2), where they learned about MINITAB, test assembly and scoring, test item banking, and CAI, all of which are now used. Although most usage is by the engineering school, other departments (especially mathematics, chemistry, and business) are becoming active users. The list has expanded each month in the last two years. Electrical engineering has also developed several microcomputers for instructional use and an HP 1000 that is used for instructional purposes and research in upper-level courses.

Instrumental in the growth of computing in engineering at this university has been an engineering accrediting agency, according to some faculty. Employment of faculty by local industry was a factor in initially creating awareness of computer potential among engineering faculty. ECMI conference attendance is credited with increasing awareness among mathematics, chemistry, and business faculty.

Political problems associated with federal efforts at encouraging integration have caused difficulties in maintaining orderly development of academic computing, according to some. Also cited as a negative factor was the pattern of federal funding which historically favored mainstream institutions and left out minority institutions. This is one of the few minority institutions, however, that managed to begin to beat the system as early as the '60s.

Conclusion

In the cases of the institutions described above, though a variety of approaches have been made toward academic computing, certain factors surface as common ingredients of success. These include campus-wide planning (or at least planning beyond the walls of a single department or class), dedication on the part of key faculty or administrators, careful budgeting practices, the ability to put together funding from various sources, the ability to learn by experience (as well as by capitalizing on the experience of others), and the will to get maximum mileage from the resources at hand. Interestingly, historical factors that could have defeated some actually seem to have caused these institutions to try harder.

References

(1) Needs/Strategy Evaluation of Minority Institutions in Educational Computing (NSF Grant No. SPI-7821515, in progress)

(2) ECMI (Educational Computing in Minority Institutions): three working conferences and one workshop for minority institution faculty in 1975-1977 to assist faculty with little or no knowledge of educational computing to learn about it in their own disciplines. Courseware was developed by a small group at one summer workshop.

(3) CMI: computer-managed instruction

(4) CAI: computer-assisted instruction

(5) CCUC: Conference on Computing in the Undergraduate Curricula (annual, 1970-1978)

(6) HumRRO: Human Resources Research Organization

(7) TUCC: Triangle Universities Computing Center

COMPUTER USE IN CHEMISTRY AT A MINORITY INSTITUTION

James D. Beck
Department of Chemistry
Virginia State University
Petersburg, Virginia 23803
(804) 520-5481

INTRODUCTION

Many students in minority institutions encounter severe difficulties in introductory science courses. These difficulties often prevent minority students from obtaining degrees in scientific fields and thus effectively shut them out of careers in the sciences, engineering, and the health sciences (1). These students are often classified as "underprepared," a classification which implies that their backgrounds, in science and mathematics especially, are not strong enough to enable them to succeed in rigorous, quantitatively oriented college courses.

A number of characteristics of these underprepared students have been identified (1,2). These range from poor mathematics background and lack of exposure to science in high school to poor study habits and a lack of self-confidence. Poor reading ability and a general weakness in communication skills appear to be significant factors which inhibit success for these students. Many of these students have low personal standards for academic achievement, expect failure, and fear science courses. Although numerous approaches have been taken to improve the performance of underprepared students in science courses, limited use has been made of computer-based learning modes.

Virginia State University is a four-year state-supported institution located in Petersburg, Virginia. About 3400 full-time students are enrolled, with over 95 percent of the undergraduate population being black. Reading tests given to incoming freshmen have shown that most of them have serious reading problems, with nearly 70 percent of them reading below the ninth grade level. Median scores on the Scholastic Aptitude Test were about 300 on the verbal part and about 350 on the quantitative part for incoming freshmen. Many incoming students have had limited exposure to science in high school. As an example, 16 percent of the students in a general chemistry class this past year had not had any chemistry in high school.

The problems of inadequate preparation are compounded by the student diversity which is encountered in many courses. Incoming freshman chemistry majors, for example, had reading scores ranging from 9.0 to 15+, SAT verbal scores ranging from 240 to 480, and SAT mathematics scores ranging from 280 to 580. This range of abilities, backgrounds, and interests presents a tremendous challenge to an instructor.

INSTRUCTIONAL APPROACH

For several years we have been experimenting with various approaches to the teaching-learning process in trying to meet the varying needs of our students who enroll in chemistry courses. Most of our efforts have been centered on our general chemistry course. This is the chemistry course with the largest enrollment and the one in which the problems of diversity and weak backgrounds are most pronounced. The approach we are currently using employs the computer in several ways (3), and we are exploring new and expanded types of computer use. In general, we try to make available to our students a wide spectrum of different ways to learn chemistry in the hope that some of these will meet the needs of all of our students. The variety of materials and methods allows students to select learning modes which suit them while permitting students who are not well-prepared to engage in extra activities to help them master the material.

Our approach has been to retain the traditional components of our general chemistry course--textbook, lectures, recitation, and laboratory sessions. To these we have added a selection of supplementary materials and activities, including a study guide, sets of performance objectives, lecture notes, slide/tape and filmstrip/tape programs, voluntary practice sessions, copies of old examinations, and suggested plans of study. The computer has also played an important role in this smorgasbord of learning activities. About fifty interactive computer programs are available for student use; most of these are drill-and-practice or simple tutorial programs. A few are simulations. The computer has also been used to generate individualized problem sets for student use. These are used off-line and the answers may be submitted for grading by the instructor.

Students are free to select which of these alternative learning strategies they wish to employ. No specific activities are required, but students are expected to complete a certain number of activities during the semester. Students are encouraged to do more than the minimum number, especially if they exhibit weaknesses in identifiable areas.

We have evaluated the effectiveness of these learning activities in two ways. One involved a simple frequency of use compilation in which we counted the number of times that students selected a particular type of activity. The other evaluation method used student surveys to ascertain the students' perceptions of the relative usefulness of the activities. Both studies produced similar results. The interactive computer programs were used heavily and were perceived by the students as being very useful to them in their efforts to learn chemistry. The computer-generated problem sets were used by nearly all students and were also perceived as being very useful. The voluntary practice sessions ranked slightly below the computer programs and problem sets in extent of use and in perceived usefulness. The slide/tape and filmstrip/tape programs were rated still lower.

It is interesting that every activity was used by some students and that every activity was rated by some to be "very useful." It is also interesting that while all of the non-traditional items that were described above were considered, on the average, to be "quite useful" to "very useful," the highest student ratings were given to the lecture notes and to the lectures. However, the textbook, study guide, recitation classes, suggested plans of study, and sets of objectives were all considered to be less useful than the computer programs, problem sets, voluntary practice sessions, and media programs. The copies of old examinations were ranked nearly even with the practice sessions. Students nearly all desired to continue having a variety of alternative methods and materials available.

COMPUTER USE

The instructional approach has been described in some detail to emphasize the integral part played by the computer. Although the computer is not an essential part of the instructional process, it is a valuable component. The computer can do some things better than others, but it cannot do all things well. It should be looked on as one of many instructional modes available to the teacher, appropriate for some students studying some types of material in some situations. When the computer is made a part of the instructional package, its use should be carefully planned with consideration given to the learning objectives that are involved.

The use of the computer in this project is not new or unique. However, these types of use can be duplicated at nearly any institution, even a small college, a minority institution, or a high school where extensive computer facilities may not be available. The methods also offer the potential of expansion to more elaborate and sophisticated computer applications.

Most of the interactive BASIC programs that we have used were written at Virginia State University and tailored to the needs of our students. While simple in concept and in construction, the programs have proven to be quite useful in this setting. They have been designed to be short and single-concept in nature, each one requiring that the student spend only a few minutes at the terminal. Most are open-ended, allowing students to obtain additional practice as they feel that they need it. In general, the programs cover basic topics which must be mastered in order to solve more complicated problems and understand more complex relationships. A few programs are simulations of chemical situations.

These interactive programs do not use graphics or special effects and generally use the most common BASIC instructions. Thus, they will run on nearly every computer system and can be used with either CRT or hard-copy terminal. Some of them have been adapted for use on several of the popular microcomputers (4). Although the programs may not be suitable for use in all environments, they are simple enough that they can easily be modified.

The generation of individualized problem sets has also been done in a very simple manner. Two different approaches have been tried here. In one we have generated the problem sets in batch-mode, using a FORTRAN program which merely selects questions at random from a question bank. Any system could use this method since a similar program could be written in any language that has a random number function. In the other method of generation, individual BASIC programs have been written for each problem set. In these the random number function is used to select individual questions and to generate individual data within questions. With this method, we can run the problem sets in batch mode or have each student obtain a copy from a terminal. Hard-copy capability is obviously required for this mode of generation.

EXTENSIONS

While the two modes of computer use that I have described are relatively simple, they can be adapted to fit a variety of instructional settings, they do not require special hardware, and they have been proven effective. There is, however, a great potential for developing more sophisticated, and perhaps more useful modes of computer use, building upon these simple applications. We have explored some interesting extensions at Virginia State University and plan to try out more new things in the future.

Prior to September 1979, our interactive programs were limited somewhat by our terminals--slow and noisy IBM 2741 hard-copy units. Installing IBM 3278 CRT units and 3287 printers has increased our capabilities tremendously. By reducing the time required to run each program, we have been able to increase the number of programs available and allow students to spend more time in remedial or advanced work. Although printers are available, students do not generally need to get hard-copy for their runs, since their results are put into a file and can be retrieved by the instructor for grading.

Although our new terminals do not have true graphics capability, their rapid print speed has enabled us to program material which requires more extensive layouts than we were able to use previously. This equipment has expanded the realm of programming possibilities to some areas that we could not consider before the change in hardware. We would like to have the capability to use real graphics; this would undoubtedly enhance learning even more (5,6).

Although we have been using computer-generated individualized problem sets for several years, we have just begun to explore the possibilities involved with using them. For example, the program that is used to generate the problem sets can also be used to generate individualized repeatable exams or quizzes. Computer grading, either on-line or off-line, is a possibility.

Problem sets offer some significant advantages over interactive programs. They do not require extensive connect time, since the actual work is done by the students off-line. They permit the inclusion of topics and problems which are not appropriate for interactive programs. For example, multi-step problems that require extensive calculations can be included on the problem sets. Questions that require the use of the textbook or of other sources of information are suitable. In fact, there are almost no limitations on the types of questions which can be included on a computer-generated problem set. If grading is not a problem, even essay questions are possible.

One exciting potential use of computer-generated problem sets is for diagnosis and prescription which we have tried on a limited basis. After the student has completed the problem set, an interactive program is accessed on a terminal. This program checks some or all of the student responses. At this point, some of the advantages of an interactive mode can be realized. Immediate feedback can tell the student which answers are correct, and the correct answers or methods of calculation can be presented. A course of action can be prescribed to help the student overcome the identified weaknesses. Prescribed activities may include reading the textbook, doing additional problems, running interactive computer programs, viewing slide/tape programs, or seeing the instructor. These prescriptions are individualized and are based on the student's performance on the problem set. In addition to providing useful information to the student, the results can be made available to the instructor, enabling areas of weakness to be identified, for individual students and for the class as a whole. This capability is similar to the TIPS program (7).

The problem sets might also serve as a core for a system of computer-managed instruction. We have been exploring some different ways to use them and have been encouraged by the results. Our students have responded positively to our primitive attempts at diagnosing problems and prescribing remedies, and we hope to enlarge this effort in the near future.

Several years of working with computers in chemistry at a minority institution have convinced me that many students, underprepared ones in particular, can benefit from computer-based instruction. I believe that this is possible without elaborate computer hardware, extensive programming experience, or a huge investment of time and money. Alfred Bork has stated that there is no one right way to use computers in education, or even a most profitable way (8). Certainly there is no one best way to use computers in science instruction or with underprepared students or at a minority institution. But there are many meaningful and worthwhile ways in which computers can be used to enhance learning for all students (9,10,11). Instructors at minority institutions should not consider computer-based instruction to be beyond their capabilities. The computer is an important instructional tool that needs to be included in planning present and future educational delivery systems.

REFERENCES

1. McDermott, Lillian C., Piternick, Leonie K., and Rosenquist, Mark L., "Helping Minority Students Succeed in Science," *Journal of College Science Teaching*, *9*, 135 (1980).
2. Kotnik, Louis J., "Teaching Science to the Disadvantaged Student in an Urban Community College," *Journal of Chemical Education*, *50*, 467 (1973).
3. Beck, James D., "Using the Computer in the Teaching of Science," *Proceedings of the Minority Institutions Curriculum Exchange Conference*, Washington, D.C., p. 37 (1979).
4. Several programs are available from Programs for Learning, Inc., P. O. Box 954, New Milford, CT 06776.
5. Bork, Alfred, "Computer Graphics in Learning," *Journal of College Science Teaching*, *9*, 141 (1980).
6. Soltzberg, Leonard J., "Computer Graphics for Chemical Education," *Journal of Chemical Education*, *56*, 644 (1979).
7. Shakhashiri, Bassam Z., "CHEM TIPS - Individualized Instruction in Undergraduate Chemistry Courses," *Journal of Chemical Education*, *52*, 588 (1975).
8. Bork, Alfred, "Computers and the Future of Education," *Computer-Based Science Instruction*, Andre Jones and Harold Weinstock, editors, NATO Advanced Study Institute, Leyden, Netherlands, p. 21 (1977).
9. Lower, Stephen, Gerhold, George, Smith, Stanley G., Johnson, K. Jeffrey, and Moore, John W., "Computer-Assisted Instruction in Chemistry," *Journal of Chemical Education*, *56*, 219 (1979).
10. Cauchon, Paul, *Chemistry with a Computer*, Programs for Learning, Inc., P.O. Box 954, New Milford, CT (1976).
11. Moore, John, Gerhold, George, Breneman, G.L., Owen, G. Scott, Butler, William, Smith, Stanley G., and Lyndrup, Mark L., "Computer-Aided Instruction with Microcomputers," *Journal of Chemical Education*, *56*, 776 (1979).

EDUCATIONAL USE OF COMPUTERS IN PUERTO RICO

Frank D. Anger
Department of Mathematics
University of Puerto Rico
Rio Piedras, Puerto Rico 00931
(809) 764-0000

ABSTRACT

Puerto Rico presents a unique blend of problems and promise in educational computing. Both the lack of an adequate technological base in the community and restricted capital hold back the development and implementation of computerized instruction in all forms. Public schools, facing many of the same problems of mainland inner-city schools, are totally unable even to start in this area. Nonetheless, since about 1974 there has been a rapid growth of instruction with and about computers in the universities, colleges, and private and parochial secondary schools. This growth promises to increase radically during the eighties.

The educational problems to which this development responds, the kinds of computer strategies attempted, some interesting results, and projections for the eighties will be discussed.

Computer Laboratories in Education

MICROCOMPUTERS IN THE TEACHING LAB*

Dr. Robert F. Tinker,
Director, Technical
Education Research
Centers, 8 Eliot St.,
Cambridge, MA 02139
(617) 547-3890

AN OVERLOOKED AREA

While computers have many uses in science education, their use in instrumentation has been largely overlooked. This paper will show the many powerful things that can be done with a properly interfaced computer to gather, analyze, and present real laboratory data.

The major reason that this educational tool has not been developed in the same way as other educational applications of computers relates to hardware. Twenty years of experiments with educational applications of computers have almost entirely involved large time-shared mainframe computers, which are not well adapted to the timing requirements imposed by real laboratory measurements. A time-shared computer determines when data is going to be read from the terminal, and so communications with it are determined by the computer's timing requirements. While that condition may usually be adequate for human interactions with a terminal, it is not adequate for most laboratory interactions. In lab applications, the timing requirements are often too stringent for simple time-sharing systems. It is possible to use time-sharing for laboratory measurements, but it is considerably more complex and expensive than using terminals for time-sharing.

The arrival of microcomputers on the scene has completely changed the hardware situation. Many laboratory applications of computers require very little computation and place only minimal requirements on the computer. As a result, very simple and inexpensive microcomputers can be successfully used in the laboratory. Now that the hardware question is being solved, it is time to begin a vigorous effort to make up for lost time, to begin using computers in the lab, to develop related hardware and software, and to research the educational implications of this tool.

SAMPLE APPLICATIONS

Our group at TERC began experimenting three years ago with applications of microcomputers in the laboratory because we were writing course material on instrumentation as part of our modular electronics project, which is designed to prepare students to use instrumentation. It was clear from the beginning that no such course would be complete without a fairly thorough coverage of microcomputer appli-

---

*Some of the materials incorporated in this work were developed with the financial support of the National Science Foundation Grant Nos.77-11116 and SED 79-06101.

cations in instrumentation; research labs that are not already heavily dependent on computers for gathering and analyzing data will certainly become so by the time the current generation of students are on the job for any length of time.

The Cooling Curve Experiment

Historically, the first application we developed used the KIM computer to record temperature of a liquid sample of naphthalene as it cooled through the crystalization temperature. This is the cooling curve experiment familiar to those who have taught introductory physical science. There is some nice physics in it because, as the sample loses heat to the surrounding water or air, the temperature does not drop uniformly. At the transition temperature, heat flows out but the temperature does not drop, giving rise to a plateau-like graph of temperature as a function of time. This shows very clearly something that many beginning students do not appreciate; namely, that there is a distinction between heat and temperature; that at the solidification temperature, heat is flowing out without a temperature change.

There is some good science in this experiment. First of all, of course, the plateau temperature is a good way of determining the melting point of a sample. In addition, mixtures of napthalene and paradichlorobenzene (moth flakes and mothballs) display some interesting melting phenomena. In some ratios, the plateau disappears, as it will for most mixtures, but at one particular ratio -- called the eutectic -- a plateau reappears at a temperature that is below the melting temperature of either of the pure substances.

All these phenomena are accessible to someone with as little instrumentation as a mercury thermometer and a stopwatch. However, gathering and plotting the data can be a tedious and boring task, particularly when it is necessary to repeat the experiment several times for different mixtures. A typical cooling curve run takes 20 to 30 minutes. This run can be speeded up by using a smaller sample, but then many fewer points are obtained and many of the important features of the cooling curve are lost.

The microcomputer solves this problem entirely. We developed a very simple interface that can be used to record the temperature of a sample at any rate desired and to display on a simple oscilloscope a graph of the resulting temperature versus time. The microcomputer can log in at a rate of one per second to generate an apparently continuous graph, which can be observed as it evolves. The temperature detector is an inexpensive signal diode. Because the transducer is small, a small sample can be used, and the experiment can be speeded up so that it only takes a few minutes. As a result, it is feasible for students to obtain data on many different samples and focus their attention on the phenomena, rather than on the boring details of gathering and displaying the results.

A Potpourri of Applications

Since this initial venture into microcomputer applications in the laboratory, we have developed over a dozen other lab applications for simple microcomputers. These are listed below with some of the more general applications described first, followed by some programs tailored to the specific needs of certain laboratory experiments:

Counter/Timer. In this program, the computer displays counts or elapsed time. It has all the functions of a normal counter/time but with a lower frequency response. In addition, it can record for later recovery the number of times that multiple events happened after an initial trigger period.

IC Testing. This convenient program learns the logic expected from an integrated circuit (IC) by running through the permutations of a functioning IC and then comparing this to questionable ones.

Function Generator. Using an analog output, this program generates a variety of functions that one would only expect from a very sophisticated function generator. Ten different functions, including white, pink, and blue noise, are available with selectable amplitude, offset, and frequency up to a bandwidth of 20 kHz.

Transient Recorder. This program triggers when the transient starts and displays the result on a standard oscilloscope, thus effectively converting it to a storage-type oscilloscope. Up to 256 samples of the input signal can be recorded at intervals as fast as 20 microseconds. In effect, the cooling curve experiment is a special adaptation of this facility.

Fourier Synthesis. This program can require the amplitude and phase of up to 30 terms in a Fourier synthesis. The resulting output waveform is displayed on a common oscilloscope and can be heard with the help of a simple power amplifier.

Fourier Analysis. This program uses the idea behind the transient recorder to capture an input waveform, which is then frequency analyzed. Using a Fast Fourier Transform algorith, the computer displays the power content of the first 128 frequencies in the captured signal.

Radioactive Half-Life Experiment. Here, pulses from a Geiger Tube are counted over time, and the resulting half-life decay function is displayed on an oscilloscope.

Pulse Height Analysis. This program uses a special interface to capture the height of pulses from a standard photo-multiplier particle detector. The resulting pulse height spectrum is then displayed on an oscilloscope. With less than a dozen integrated circuits in the interface, a standard microcomputer can be used to replace special-purpose PHA instruments costing ten times as much.

The Computer of Average Transients. We have developed a geological sounding experiment that requries careful processing of signals from a geophone that detects seismic waves generated by hitting the ground. The common technique is to use a computer of average transients, which the KIM simulates, using a simple program.

Solar Collector Analysis. Temperature from a number of sensors distributed around a solar collector, as well as a light detector measuring the input light levels, is simultaneously logged and selectively displayed through use of this program.

Rotational Dynamics. This program, written by Ken Flowers, measures the angular acceleration of a disk commonly called for in lab experiments involving rotational dynamics.

Linear Dynamics. An electronic version of spark tape uses clear tape with black lines. When attached to an object and pulled through a special detector, the computer can display position, speed, and acceleration instantly.

EDUCATIONAL IMPLICATIONS

These examples illustrate that it is technically and economically possible to have students use the computer as a general-purpose laboratory instrument; the big question in many people's minds is whether such use is desirable. I will begin addressing this point by first developing the skeptics' argument more fully. There are essentially three main arguments against the use of this kind of sophisticated instrumentation. First is that it is too complex and difficult to use; secondly, it makes the laboratory mysterious, thus weakening the connection between reality and result; and third, it isn't really necessary at all. These arguments are taken up in order below.

Complexity

Microcomputers introduce a whole new level of complexity that requires mastering sophisticated hardware, software, and operating concepts. In order to use computers effectively, the students will have to investigate all these new ideas and will get sidetracked, so that any possible advantages that the computer can bring will be offset by the enormous investment of time and intellectual effort required.

This would certainly be a compelling argument if it were true, if students were required essentially to develop all the hardware and software needed in the laboratory. However, this is an inappropriate way to use a computer in the laboratory. The desirable approach is to use previously developed canned programs that are carefully designed to require the minimum amount of additional knowledge. Lab programs should operate just the way electronic pinball machines do; anybody with a quarter can walk off the street and begin using those microprocessor-based machines. There is no reason that laboratory instruments using microprocessors should be any more difficult to use. Admittedly, some of our first efforts in this area do not meet this requirement. But there is no basic reason that prevents microcomputers from being easy to use in an application area. It is simply a question of good programming; providing menus for selection and help files to clarify any difficult points.

Mystification

The skeptic would hold that an instrument that instantly gives results is undesirable when it is not clear to students how those results are obtained. If there is no alternative way to cross-check the results, if the experiment must stop when the apparatus stops functioning, and if the computations are too difficult to discuss, then the results may as well be magic.

Since the results appear to be magical, the student loses any chance of having any intuitive understanding of the meaning of the results and connection between those results and the physical phenomena under investigation. For instance, you pull a strip of marked paper through a little gadget, and the computer prints out some acceleration data. The connection between the way you pull the tape and the acceleration is incomprehensible and unverifiable, and, therefore, the students' understanding of the term acceleration is not enhanced by this experiment. With a spark tape, the student can see the marks on the tape, understand the relationship between those and the motion, and trace the calculation of the acceleration through a series of simple calculations based on the marks. Furthermore, the calculations are based directly on the definitions of velocity and acceleration, and therefore, the student gets practice in applying those definitions by performing the calculations in the laboratory. All awareness of these relationships are lost when the computer automatically calculates the accelerations and graphs them.

An important part of laboratory experiences for students is their enabling function. That is, by learning how to frame and answer questions, students are then better able to function as scientists and as citizens. Understanding is the key to this enabling function, and the inclusion of a major thing, the microcomputer, which is inherently incomprehensible, is disabling rather than enabling.

These are important arguments and point up some of the pitfalls of using the computer in the laboratory. Like any new tool, the microcomputer can be misused. However, the statement of the problem points to the solution. Most people can learn to use mysterious or incomprehensible devices -- so long as they have a clear idea of cause and effect. Many aspects of driving a car are essentially mysterious to drivers: the details of the connection between the wheel and the direction of motion, between turning the key and the operation of the motor, between the position of the accelerator and speed. The same arguments can be applied to most of the technologies in our environment: television, radio, calculators, telephones, and aspirin. People use them all more or less effectively without any detailed understanding of their mechanisms. Through practice people have learned the connection between inputs and outputs. They have learned what to expect when the accelerator is pressed, when the record button on a tape recorder is punched, when a telephone is dialed. The reproducible and predictable nature of these technological aids, which can be understood at an intuitive level through practice, completely removes the mystery surrounding their operation and obviates the need for a detailed understanding of how they work. This process of gaining an intuitive appreciation of a connection of inputs and outputs I call "intuition calibration" because it is directly analogous to the calibration one performs on instruments.

The thesis, then, is that if a microcomputer is used to measure acceleration, this will appear to be mysterious only until the students' intuition is calibrated. How is this done? The direct approach is to give students this tool and let them play around with it in a directed manner in order to get a good intuitive feel for what it is producing. In tests with students, we have found that it takes very little time using this tool to appreciate, on an intuitive level, the relationship between the acceleration and the change of speed, namely, that rapid changes in speed results in large acceleration and that slowing down is just another form of acceleration with a negative sign. These intuitive explorations demystify the tool, but more important, they foster an intuitive feeling that no other approach can create. Students come away with more than understanding of an isolated term, such as acceleration. They talk about acceleration in intuitive terms that are directly related to its definition. They see that it is position-independent, that it only depends on how quickly speed is changing. Thus, the measurement and the definition come together in the laboratory without the use of computation.

The one argument that can't be countered is that students do not get as much practice performing computations in a laboratory where the computer does the computation. The obvious cure for this is to assign some additional computation or to do the experiment once without the computer.

Necessity

The skeptic's argument here is that the computer, while attractive, is not really necessary in the laboratory. However, the microcomputer makes a number of things possible that could not otherwise be done conveniently or economically, and therefore, unless one takes the fantastic position that there is no need to improve science instruction, one cannot ignore these rather substantial improvements.

The use of a general-purpose laboratory computer broadens the kinds of experiments that can be contemplated in a teaching laboratory and also increases the rate at which data can be gathered and analyzed. For example, the dynamics measurements permit a detailed analysis of the connection between force and acceleration that is simply not possible through any other mechanism. This application alone opens up new and, as yet, uncharted possibilities in the introductory laboratory. Similarly, the Fourier Analyzer, the Pulse Height Analyzer, and even the Computer/Timer open up possibilities that would otherwise not be considered in elementary laboratories because of cost considerations.

The speed with which experiments can be analyzed is often an important educational factor. Instead of a simple falling ball experiment occupying an entire laboratory, many accelerated motions can be studied together in one laboratory; instead of one cooling curve, a set of cooling curves of various mixtures can be studied in the same length of time. The result has to be that students will have more raw data available to them on which to fashion their theoretical model of the natural world. Because less time is devoted to the calculation and presentation of data, more time can be devoted to understanding the scientifice phenomena. In many cases, the hand calculations are not only time-consuming but distracting; because they occupy most of the lab time, students tend to focus on them and totally forget the scientific phenomena being investigated.

A secondary but important argument for the inclusion of microcomputers in the teaching laboratory is that by the time our students are employed, those that go into the laboratory will find microcomputers there.

THE FUTURE

I feel that the microcomputer viewed as a laboratory instrument is an emerging resource which is cost effective today in certain teaching situations and will be used in the immediate future in an extremely broad range of science teaching environments.

Current Efforts

The primary impediment to the wide scale use of microcomputers in the laboratory now is that software and hardware are not readily available from any one source. This, in turn, is partly because commercial vendors do not perceive the teaching laboratory as a commercially viable area because there has been no broad expression of interest by laboratory teachers.

As a non-profit organization dedicated to improving science instruction, our group at TERC have undertaken a number of projects to speed the introduction of the microcomputer into the laboratories. We have, of course, generated some sample programs that illustrate the kinds of things that can be done, and we also evaluate and distribute hardware and software so that schools and teachers can find most what they need at one place. We have developed three laboratory interfaces and are currently working to interest some manufacturers in these or other related products.

For a number of years, we have been giving workshops for teachers on the use of microcomputers in teaching. These workshops have been very well received and, in fact, we face the problem that there are more people interested in taking these workshops that we can possibly enroll. To meet this need, we are in the process of developing an exportable workshop -- a one-day workshop that former workshop participants can themselves offer to others. We are currently looking for groups of teachers who will participate in the first round of these workshops and be willing, then, to host these workshops themselves in their immediate locale.

Long-range Efforts

We see a time in the near future when microcomputer-based instrumentation will be as widely used in introductory science teaching as microscopes, clocks, and thermometers. The microcomputer will not be a frill added on top of other instrumentation; it will be the primary instrument. Because of the ease of programming and flexibility of input and transducers, grade school students could use microcomputers in much the way they currently use microscopes, to extend their senses, to help them perceive physical phenomena that are outside the range of their immediate senses, and to provide the questions and motivations for scientific studies. Most of the elements necessary to fulfill this vision are currently available, and it is certain that prices will drop over time to the point at which few schools could justify not having microcomputers in the laboratory.

Meanwhile, however, we see four areas in which important work needs to be done before microcomputers will be widely used: education research and development, improvements in the laboratory interface hardware, network software for sharing limited resources, and development of curricula that use laboratory-interfaced computers.

Education Research and Development. We have suggested that the introduction of microcomputers in the laboratories, while having many advantages, also creates certain pitfalls. Both the advantages and the pitfalls need careful researching. In what sense can students learn to master such a powerful tool? What intellectual prerequisites must they have? The questions raised about mystification and enabling of students must be studied in greater detail. There is a possibility that the introduction of laboratory experiences that use the microcomputer will be of benefit not only to science instruction but also to mathematics learning. When combined with the power of the computer to analyze and model, the introduction into mathematics instruction of computers that can deal with real phenomena will have an important role in motivating students in building intuition; this area needs to be researched and developed.

Interface Hardware. We see a continuing need to keep abreast of hardware developments and continually incorporate relevant innovations into interface designs. It is only by producing and distributing increasingly sophisticated prototypes that we will learn how these can be used in education, how costs can be minimized, and how dissemination problems can be reduced. One of the benefits of our work in building and distributing prototypes is that it helps build a market we hope commercial companies will want to exploit.

We see a clear need to develop an interface that contains its own 16-bit CPU and communicates to a host computer over the new general interface bus or over other standard communication lines. Such a device would have greatly increased performance and would be compatible with almost all commercially available computers. While it may not be affordable by a large number of institutions this time, it is important to learn, through limited distribution, what educational advantages accrue from the increased performance this unit would have.

Networking. The expensive part of computers now is not the central processing unit or the memory but rather the peripherals: printers, disks, specialized interfaces, graphic displays, and the like. The appropriate way to maximize computational performance is to have a means of sharing the expensive resources among a variety of inexpensive computers. This is called networking and is the microcomputer alternative to time-sharing which is really only appropriate for computers that have expensive CPUs and memory. We have some interesting networking software and hardware currently operating in our lab that need to be brought into the teaching environment and expanded to laboratory applications. We envision that the teaching laboratory of the future will involve a network of inexpensive computers at each laboratory station, networked with a number of more sophisticated special-purpose microcomputers to be used for extensive data analysis, combining data from many students and generating complex displays. This local network would be tied into a central computing center, which could be used for long-term storage, high-speed printing, and the maintenance of large data bases. We are cooperating with a number of schools to work towards an initial implementation of this ideal.

Curriculum. The most pressing current need in this general area is to develop curriculum material that uses laboratory microcomputers. One can view the programs that we have developed so far as samples, a shotgun blast that illustrates what can be done. There is an urgent need systematically to apply laboratory-based microcomputers to courses in various disciplines. An important aspect of this effort is to try to define the electrical characteristics of the interface that is required, so that the curriculum material does not have to be tied to any particular hardware. We naturally feel that our lab interface defines this standard, but we would be delighted to discuss any other possible standards with any educators.

The only way extensive curricula will be developed is to use teacher-generated material. The major problem with this approach is the evaluation and distribution of this material. We have established an experimental Microcomputer Teacher Resource Center that uses a commercial data base to store reviewed files and programs. This data base is public and contributions are welcomed. If teachers will evaluate what they use from this data base and contribute some new material, great strides will be made in filling the missing curriculum gaps.

SUMMARY

A microcomputer equipped with a laboratory interface offers a very exciting new resource for science instruction. With the proper hardware, the computer can be turned into an altogether general-purpose instrument. It can replace a large number of standard instruments at a fraction of the cost, and provide the major measurement and analytic tools needed in a wide range of science courses. Equipment and software are available today that begin to do this, while future hardware developments and declining costs will make lab-based microcomputers a necessity in many teaching situations.

THE COMPUTER LAB OF THE 80S

Guy Larry Brown
Head, Data Processing
Piedmont Virginia Community College
Rt. 6, Box 1-A
Charlottesville, Virginia 22901
(804) 977-3900

INTRODUCTION

During the past 20 or so years that the computer has been used as an academic tool, there has been considerable discussion about computer-assisted instruction (CAI). CAI, in fact, has been a luxury affordable by only a few. In recent years, with pressure being applied to educational budgets, it appeared that CAI would remain a dream to be realized only if the good fairy godmother waved her magic wand in a classroom. However, the advent of the microcomputer with its low cost and super capabilities has made it possible for that marvelous wand to be directed into even the most impoverished academic niches. This paper describes an operating system of microcomputers that is unique in capabilities and costs.

THE SYSTEM

The system in operation consists of 14 microcomputers each with 8,192 (8K) of random access memory (RAM) and 8K of read only memory (ROM). The RAM allows storage of instructions written in Microsoft BASIC and data in the form of letters of the alphabet, digits, and special characters. The ROM contains an interpreter that translates BASIC into the computer's language. Each micro has its own keyboard for input into the computer and a video monitor for visual output.

Stored programs are available from shared dual floppy disks, each capable of holding up to 275K characters. The disk unit is part of the host computer through which control of the micros is centrally exercised. This host computer has 48K RAM with its BASIC on disk instead of in ROM. Control of the host is through a CRT. Attached to the host is a 100 character per second (CPS) matrix printer which is also shared by the micros.

The program (software) that provides the capability for the micros to share the disks for input/output (I/O) and the printer for hardcopy output occupy only a small part of a single disk. The remainder is set aside to store student programs and for other uses as may be desired. For example, a program can be called by a student to provide information about a problem he does not fully understand. Or a series of tests can be stored for use by the student at the instructor's discretion.

Two of the micros received minor modifications to permit I/O through a standard cassette player. This capability permits the units to be detached from the system and moved out of the lab for demonstrations or other purposes. It also allows for the transfer of a program stored on a cassette into the micro's memory and hence onto a disk for storage and subsequent availability to all micros.

All computers were manufactured by Ohio Scientific, the printer is a Centronics 779, the video monitors are Sanyo VM 4209, and the CRT is a TEC Series 500. The system is capable of operating with 16 of the micros; however, to keep total costs below $19,000, only 14 were purchased initially. This figure included all equipment, installation, and software! Equipment selection was influenced by the availability of a reputable local dealer who services what he sells. Also favoring Ohio Scientific is the large inventory of software, a small business system capability with hard disks, and design features making expansion and upgrading simple and relatively inexpensive.

OPERATION

The system has operated almost flawlessly since installation in the fall of 1979. Typical micro systems use cassette I/O which is very slow compared to disk. And a dedicated disk for stand-alone micros adds significantly to the cost of the unit. The shared system allows each micro user to (1) load a program stored on disk into the memory of his computer, (2) save a program on disk, (3) print a listing of the program, or (4) print the output from the execution of a program. Only one micro can use the disk or printer at a time. The use of passwords for files, message sending to the host computer, and line advancing on the printer are recent, locally added enhancements to the operating software.

USES

Presently the system is used only for teaching the programming language BASIC. One 20-student class was taught in the fall of 1979, and two such classes are being conducted during the winter quarter.

During registration, several units were moved from the lab for demonstration purposes. Curiosity was aroused in many who stopped to engage in conversation with the computer or play a game.

FUTURE PLANS

In addition to expansion to 16 micros, there are plans for the acquisition of a quality character printer for word processing. This capability will permit instruction of secretarial science students and members of the staff in word processing.

Expansion of courses to cover assembly language is anticipated as well as a course in the operation of small business computer systems. Longer range plans call for teaching COBOL.

Efforts are being made to make the lab available to any of our faculty for use in their courses. Programs will be developed for math, chemistry, biology, and a variety of business courses.

ADVANTAGES

Cost.
Reliability.
Graphics capabilities of 256 characters.
Standardization of components.
A single disk for program/data storage.
Display of 64 characters/32 lines on micro videos.
Extensive software available from manufacturer.
Expansion of capabilities and updating inexpensive.
Not dependent upon trained computer personnel.
Rapid transfer of programs/data through a disk system.
Hardcopy available to all micros.
Control is facilitated through the host computer.
No reliance on telephone connections.
Students learn on a micro frequently found in businesses.
Micros still function independently.

DISADVANTAGES

Disk and printer allow only one user at a time (this is only a minor inconvenience).
Cost is so low that academic departments can afford procurement without involvement of computer management (maybe this is an advantage).
Documentation is not complete or error free.
Operating software lacked capability for printing the execution output of a program (locally modified to acquire this capability).
Service may not be locally or quickly available.
Cost is so low that one cannot expect the same level of manufacturer support as previously experienced when systems were priced much higher.
Demand for usage might be so high that the establishment of priorities for use may be difficult.

SUMMARY

A computer lab for the 80s is available today. It consists of up to 16 microcomputers, each of which can have up to 32K RAM. They have rapid access to programs and data through a shared dual floppy disk drive which also provides for equally rapid storage. Hardcopy listings of programs and output from the execution of those programs are also available on a shared printer. The system, including the host computer and all other hardware, installation, and necessary software is available for around $20,000.

THE EDUCATIONAL TECHNOLOGY CENTER

Alfred Bork, Stephen Franklin, and Barry Kurtz
Educational Technology Center
University of California
Irvine, California 92717
(714) 833-6911

This paper reports on the formation of the Educational Technology Center at the University of California, Irvine. The primary focus of the Center is the use of the computer as a learning aid. The Educational Technology Center was started on January 1, 1980, with University funds providing staff support. The Center continues the activities in computer-based learning conducted by the Physics Computer Development Project during the last eleven years.

NEED

The Educational Technology Center was formed because we believe strongly that the next decade will be a critical period in American education. Such centers are needed to guide us toward a future where the computer will play an extremely important role in education. It is important to develop a number of continuing groups that are not fully dependent on grant funds but have an existence beyond support for particular projects.

We have pursued for some years within the University of California the possibility of one such Center. We will provide guidance to others working in this area. The Center will work on a wide range of research and development activities leading to more effective use of the computer and associated technologies in learning environments.

CURRENT ACTIVITIES

The Educational Technology Center intends to engage in many activities concerning more effective and more efficient use of information technology in learning, emphasizing learning materials on the personal computer. Some of the activities will be pure research, while others will have a strong applied and developmental component. We shall work closely with individuals and groups elsewhere, as in the past, so that the Center has a nationwide effect beyond its immediate activities, materials, and publicity.

The Center will publish a newsletter reviewing the activities and results of its projects. Although no set schedule is planned, we expect this newsletter to be published three times a year. Anyone interested in receiving the newsletter should write to the Center.

The following list gives the active projects at the Irvine Center. Further information about any activity is available on request.

1. A Testing and Tutoring Environment for Large Science Courses.
   Authoring for personal computers
   Testing environments
   Physics - waves
   Statistics
   National Science Foundation--Comprehensive Assistance to Undergraduate Science Education (CAUSE)
2. Scientific Literacy in the Public Library.
   Public libraries, shopping centers,
   science museums
   Public understanding of science
   Personal computers
   Fund for the Improvement of Postsecondary Education (FIPSE)
3. Mathematics Competency Tests for Beginning Science Courses.
   University of California/California State University and Colleges
4. Translation of timesharing materials to personal computers.
   University of California/California State University and Colleges
5. Biology materials - ecology.
   University of California, Irvine, Committee on Instructional Development

6. Development of Reasoning Skills in
   Early Adolescence.
   Junior high students
   Transition to formal reasoning
   Personal computers
   National Science Foundation - Developments in Science Education (DISE)

PRODUCTION SYSTEM

In addition to specific products, such as those just mentioned, the Center has developed a production system for generating computer-based learning material. The emphasis is on both efficiency and effectiveness and on techniques which will allow natural extensions to large-scale production of such models. The production system is based on a systems analysis of the problem and on our many years of experience in producing a wide range of learning material. Literature is available describing the system and the supporting software.

ISSUES FOR THE FUTURE

Currently we can distinguish a number of very important issues that will shape the future of computer-based learning; these issues indicate directions the Educational Technology Center will pursue. No order of priority is intended in this list.

1. Full-scale course development. At present, with a few notable exceptions, computer-based learning materials are supplementary to course structures. Very few full courses make heavy use of computers to aid learning. We need experience in developing such complete courses and in integrating computer and other learning aids. We need additional experience in computer-aided delivery of such courses.

2. Expanded acquaintance. Very few teachers, and even fewer members of the general public, have seen any effective computer-based learning material. Often the examples seen have been weak examples; so the learners have formed inaccurate opinions of the value of such material. We need more acquaintance with the full range of possibilities, more computer literacy with a learning emphasis.

3. Research in learning. Presently we have conflicting theories about learning. We need to know more about how students learn so that we can develop better learning aids.

4. Production techniques. Older strategies for developing materials often were not suited for the large-scale development needed in the years ahead. The types of systems approach followed at Irvine and elsewhere needs further exploration and refinement as the scale of activities increases. We should aim for the best possible materials at the least developmental cost.

5. Expanding technologies. Computer and associated technologies are evolving rapidly. We must learn quickly to use an expanding range of capability, developing materials which are not immediately outmoded.

6. The computer as a new interactive medium. In understanding a new learning medium, we must learn how it differs from older media. For example, reading from computer displays has many differences from reading print medium, but the empirical details are not known.

7. Dissemination. New media also demand new modes of dissemination.

8. New course and institutional structures. As computers are more widely used, they will have major effects on course and institutional structures.

The Educational Technology Center intends to pursue these and other issues.

Invited Session

MIS EDUCATION:
INDUSTRY NEEDS AND EDUCATIONAL SOLUTIONS

Chaired By Eleanor W. Jordan
Department of General Business
Business-Economics Bldg. 600
University of Texas at Austin
Austin, Texas 78712
(512) 471-3322

ABSTRACT

Over the past decade a considerable number of discussions in industry-oriented publications like Computerworld have focused on irrelevant education as a reason for prevalent software problems and DP personnel shortages. In this session, educators will discuss efforts made by their institutions to resolve the supposed relevance problem in programs for business application's software designers at the graduate and undergraduate level. Two industry representatives will also participate in the panel.

PARTICIPANTS (Listed in order of Presentation

Marguerite Summers, Chairperson
Western Illinois University

David Naumann
University of Minnesota
MBA and PhD MIS programs

Joyce J. Elam
Wharton School
University of Pennsylvania
MBA concentration in MIS

Eleanor W. Jordan
University of Texas at Austin
Undergraduate DP program

James Cook
Southwest Texas University
Undergraduate business CS program

Ken Truitt
ARCO Oil and Gas Company

Willis Ware
Rand Corporation

Computer Games in Instruction

SHALL WE TEACH STRUCTURED PROGRAMMING TO CHILDREN?

Jacques E. LaFrance
Dept. of Mathematical Science
Oral Roberts Univ.
Tulsa, OK 74171

ABSTRACT

Pre-college programming experience has been observed to be detrimental in many cases to college-level study of computer science. The problem is the lack of understanding of the principles of structured programming. The solution proposed here is the introduction of structured programming games at the elementary school level which will prepare the children to do structured programming later on with languages such as BASIC and FORTRAN that are not designed to promote structured programming. An example of doing this called ANTFARM is presented and the results of using it with one group of children are discussed.

THE PROBLEM

More and more students are entering university computer science programs with some form of prior experience with computers. This is typically a high school course in FORTRAN or BASIC or experience with a personal computer. There seems to be no corresponding exposure to structured programming or design, however. The students believe they know a lot because they have written programs in BASIC or FORTRAN, but in reality they know only coding; they understand nothing of structure, top down design, good style, or documentation. It is sometimes difficult to get them to break the resulting bad programming habits. This problem is likely to be compounded by the increasing availability of small personal computers without proper accompanying instruction in good top down programming. An increasing number of young people will likely learn programming by reading their microcomputer BASIC manuals. The learning of coding will thus be encouraged rather than the learning of programming. Yet these students will be misled into thinking they know a lot because they are able to code some substantial programs. The task of computer science education will be made more difficult by this background experience because the students will have to do more unlearning than is currently the case and sometimes unlearning is more difficult than learning.

THE PROPOSAL

A possible solution to the above problem would be to introduce structured programming concepts to children before they are able to begin to use conventional programming language. Other solutions might be to replace BASIC with PASCAL as the most common microcomputer language or to have all manufacturers' BASIC manuals based on structured programming. Neither of these alternative solutions seems feasible. The simplicity of BASIC will cause it to continue to be popular, especially with the smallest computer configurations. The large number of programs already coded in some dialect of BASIC will also serve to keep support for BASIC strong. The manufacturers may be more or less willing to include structured programming concepts in their manuals, but the bottom line will always be what enables them to sell their product. The general public will have to become more knowledgeable, discriminating, and particular before the manufacturers will feel any serious pressure to include material on structured programming.

Although the first solution mentioned above is not a shoo-in, it is one that can be developed more easily than either of the others and would be worth investigating. It is not realistic to have all elementary schools begin teaching structured programming, but a few well-published success stories plus the availability of well-written materials for use in schools would help to promote the idea. A few successful projects at schools in several areas across the country would encourage other educators to do something similar and inspire computer manufacturers to include structured design and programming in their manuals.

Structured programming concepts could easily be introduced to elementary school children if they were presented in a game form at their level of sophistication. Games could be developed for which the control language is inherently structured, and the children would be motivated by the attraction of the game to master the control language as well as possible in order to do more with the game. Because the language would be inherently structured, they would automatically begin to develop structured problem solving skills and structured expression of their ideas. Manuals could be provided that would describe structured design and programming in a way that most teachers could follow and would also describe how to use the games with the children to develop their understanding and skills.

The LOGO system at M.I.T. Artificial Intelligence Laboratory (12) already contributes to these goals. This system was not developed for introducing structured programming concepts but could be easily adapted to that purpose. It is exciting for the children to use (14) and therefore has the necessary motivational characteristics. Several other tools for teaching structured programming could be developed with different appeals, different levels of sophistication, and different organizational structures.

THE EXPERIMENT

A pilot testing of these ideas was made in April 1979 by the author and Dr. Mary Dee Fosberg of Central State University, Edmond, Oklahoma. To present the ideas of programming to a group of gifted children, we designed a game called ANTFARM. It was implemented on an IMSAI with a Z80 processor, Digital Microsystems disk, and Infoton Z00 CRT using the UCSD PASCAL system which we took to the presentation and set up there.

The ANTFARM program consists of drawing an ant facing up in the center of the screen and two rows of food ("@"'s) in the upper left. The goal is to have the ant move over to the food, eat it, and plant new food. The ant looks like:

```
                    *1                   \  /
 \*/                                     *00-
  0       or        -0        or         /  \
  0                   0-
 /1\                  1\
```

etc., depending on orientation

It accepts five basic commands: MOVE, TURN LEFT, TURN RIGHT, EAT, and PLANT. MOVE means move forward one character position in the direction the ant is facing; TURN, LEFT or RIGHT, means a 45° rotation about the first body segment; EAT means to consume whatever is under his head; and PLANT means to plant a new seed with his tail. Seeds start out as "."; 100 time units later they germinate (","); then after 50 more units they sprout (";"). They grow into a stalk ("1") after 50 more units and then into a branching plant ("Y") after 50 more. At the end of 75 more units, a flower ("P") appears and then after 75 more or a total of 400 units, the plant matures into food ("@"). One time unit is the time necessary to execute one operation. Such a growing rate seems to be a satisfactory choice though unrealistically fast.

Each time the screen is updated, the cursor is moved to the home position, and the first line on the screen is erased to await the next input. The user may type any combination of commands on this line. When the return key is pressed the commands are performed. We began by simply showing the children the effects of individual commands. Then we introduced the concept of sequence by showing that several commands can be listed on the same line.

Since the sequence of commands cannot be longer than one line, the limit on program size is quickly reached. Other commands are then introduced to allow the ant to do more things. From the author's point of view, however, additional commands are given to develop the additional structured programming concepts of iteration, selection, and refinement.

The first additional command illustrates iteration: "DO a command sequence n TIMES," where n is an integer. Instead of "n TIMES," there can be "TO ROW integer" or "TO COLUMN integer." The rows are numbered down the left of the screen and the columns across the top on the second row. This command allows the ant to go through a sequence many times before stopping for the entering of the next command. (The "DO" command can be used in conjunction with the do nothing commands "WAIT" and "REST" to watch plants grow.)

The command to illustrate selection has the form:

$$\text{IF SEE} \left\{\begin{matrix}\text{PLANT}\\ \text{FOOD}\\ \text{SPROUT}\\ \text{DIRT}\end{matrix}\right\} \left\{\begin{matrix}\text{AHEAD}\\ \text{LEFT}\\ \text{RIGHT}\end{matrix}\right\} \text{ command.}$$

If the character space in front (or to front left or front right) contains non-space for "PLANT," "@" for "FOOD," ";" or "l" for "SPROUT," or space for "DIRT," the single command following will be executed; otherwise it will be skipped. This command allows for some interesting conditional tasks such as having the ant look for a row of food to eat or an open furrow in which to plant more food. Although the children were able to use this command it seems to need more development to be stronger. In its present form, it seems too restrictive.

To encourage the development of hierarchical structure and refinement, the command sequences are limited to one line, and a command called "REMEMBER" is given. This command has the form "REMEMBER name command sequence END" which allows the child to give names to command sequences for subsequent use as commands themselves. One popular sequence is

```
REMEMBER TURNAROUND DO TURN LEFT 4 TIMES
     END
```

after which "TURNAROUND" can be used as a command. This subprogram capability is especially helpful when used in connection with the selection command since only one command may be selected or skipped. The commands "QUIT" and "STOP" allow the current subprogram to be ended prematurely (or the ANTFARM program itself to end if it is used at the command line level). This capability allows for powerful search features when used in submodules with selection and iteration commands, such as

```
REMEMBER MOVETOFOOD DO MOVE IF SEE FOOD
     AHEAD STOP 70 TIMES END
```

This will cause the ant to move forward 70 times or until there is food right in front of it, whichever comes first.

One of the tasks the children faced was the planting of a field of four rows of ten plants each. The following is an example of a structured development of a solution:

The final commands available are "FORGET name," which eliminates the named module from the table, and "TELL," which lists on the screen all the "REMEMBER"ed names and their definitions. When the user pushes the return key after the "TELL" display, the screen is cleared and the ant's farm scene is redisplayed as it was prior to the "TELL" display. A future addition will be commands to save and recall the set of defined modules to and from a disk file. These commands will allow a child to build his program from day to day rather than having to type everything over from the beginning at each session.

THE RESULTS

We had about 1½ hours with a group of gifted children between the ages of nine and twelve. Interest was universally high throughout the session. The format of the session was that for about 20 to 30 minutes the children were shown some of the hardware aspects of computers; then the ANTFARM was introduced. Since the Infoton has a detached keyboard, the author sat to the side and did all the typing while the children looked at the screen and said what to type. We introduced the ANTFARM features roughly in the same order as in this paper, stopping frequently to ask the children how to get the ant to do some specific task. The children picked up the ideas very quickly and were soon telling us of things they wanted to see the ant do. By the end of the 1½ hours, three or four of the most involved children were beginning to use top-down design to achieve some specific task they wanted the ant to do, defining their own modules and giving them names.

The only weaknesses observed were the selection commands and the bugs. The selection commands were too difficult for the children to use unaided in their present form, primarily because the condition was something happening out in front of the ant instead of right under him, and the seeing left or right does not correspond well to where the ant is facing when he turns. The bugs occasion-

```
REMEMBER FIELD TWOFURROWS TWOFURROWS END
REMEMBER TWOFURROWS FURROW NEXTRIGHT FURROW NEXTLEFT END
REMEMBER FURROW DO MOVE PLANT 10 TIMES END
REMEMBER NEXTRIGHT DO TURN RIGHT 3 TIMES MOVE TURN RIGHT MOVE MOVE END
REMEMBER NEXTLEFT DO TURN LEFT 3 TIMES MOVE TURN LEFT MOVE MOVE END
```

A field could be planted in the upper right part of the screen by

```
DO MOVE TO ROW 3 TURN RIGHT TURN RIGHT DO MOVE TO COLUMN 65 FIELD
```

ally caused the program to abort, losing all the defined modules and all the farm development to that point. Actually the program worked quite well considering it was developed and implemented in three or four man-days and consists of over 400 lines of PASCAL code. (This accomplishment in itself is a credit to the value of top-down design, structured programming, and PASCAL.)

THE CONCLUSION

The ANTFARM program has demonstrated that it is possible to create tools with high motivational value for children that contain all the concepts of structured programming. Furthermore, the way this program is designed forces the children to use hierarchical or structured development to achieve their goals. Even in one session we were beginning to see some of the children using structured development on their own. It seems reasonable to suggest that continued use of tools such as ANTFARM over an extended period of time could develop these concepts so well in children that they would continue to use top-down design and structured programming even with languages such as BASIC and FORTRAN which are not naturally oriented toward them. We hope that we will have the opportunity to continue to explore these ideas and that others will have similar opportunities. We need long-term data on children using these tools to determine the effects on their future success in programming or computer science.

REFERENCES

1. Banet, Bernard, "Computers and Early Learning," Creative Computing, #4-5 (Sept.-Oct. 1978), pp. 90-95.
2. Beyer, Kathleen, and Stuart Milner, "Elementary School Curriculum Task Group," ESSS Report, September 1979, pp. 23-4.
3. Brady, J. M., and R. B. Emanuel, "An Experiment in Teaching Strategic Thinking," Creative Computing, #4-6 (Nov.-Dec. '78), pp. 106-109.
4. Cohen, Harvey A., "Oznaki and Beyond," Proceedings of the National Educational Computing Conference, 1979, pp. 170-178.
5. Dahl, O. J., E. W. Dijkstra, and C. A. R. Hoare, Structured Programming, New York: Academic Press, 1972.
6. Hakansson, Joyce, and Leslie Roach, "A Dozen Apples for the Classroom," Creative Computing, #5-9 (Sept. 1979), pp. 52-54.
7. Larsen, Sally Greenwood, "Kids and Computers: The Future Is Today," Creative Computing, #5-9 (Sept. 1979), pp. 58-60.
8. Lieberman, Henry, "The TV Turtle: A LOGO Graphics System for Raster Displays," MIT, A. I. Memo 361 (June 1976).
9. MECC, "Computer Literacy Objectives from MECC," ACM SIGCUE Bulletin, #13-4 (Oct. 1979).
10. McGowan, Clement L., and John R. Kelly, Top-Down Structured Programming Techniques, New York: Petrocelli/Charter, 1975.
11. Milner, Stuart D., "An Analysis of Computer Education Needs for K-12 Teachers," Proceedings of the National Educational Computing Conference, 1979, pp. 27-30.
12. Papert, Seymour, "Teaching Children Thinking," MIT, Artificial Intelligence Memo 247 (Oct. 1971).

13. Papert, Seymour and Cynthia Solomon, "Twenty Things to Do with a Computer," MIT, A. I. Memo 248 (June 1971).

14. Papert, Seymour and Harold Abelson, Jeanne Bamberger, Andrea diSessay, Sylvia Weir, "Interim Report of the LOGO Project in the Brookline Public Schools: An Assessment and Documentation of a Children's Computer Laboratory," MIT, A. I. Memo 484, (June 1978).

15. Perlman, Radia, "Using Computer Technology to Provide a Creative Learning Environment for Preschool Children," MIT, A. I. Memo 360 (May 1976).

16. Ragsdale, Ronald G., "A Program Package for Introducing the Top-Down Approach to Computer Programming," *SIGCSE Bulletin*, #11-1 (Feb. 1979), pp. 113-117.

17. Solomon, Cynthia J. and Seymour Papert, "A Case Study of a Young Child Doing Turtle Graphics in LOGO," MIT, Artificial Intelligence Memo 375 (July 1976).

18. Watt, Daniel H., "A Comparison of the Problem-Solving Styles of Two Students Learning LOGO: A Computer Language for Children," *Proceedings of the National Educational Computing Conference*, 1979, pp. 255-260.

19. Weinberg, Gerald M., *The Psychology of Computer Programming*, New York: Van Nostrand Reinhold, 1971.

STRUCTURED GAMING: PLAY AND WORK IN HIGH SCHOOL COMPUTER SCIENCE

J. M. Moshell, G. W. Amann
(The University of Tennessee)

W. E. Baird
(West High School)
Knoxville, Tennessee

PROLOGUE

Question 1: When is a computer game not a game? When is it an ok class-activity?

Question 2: So what's wrong with games, anyway?

Exasperated answer to 2:
Students won't work on programs when they have access to games. Games are *fun* and programs are *work*.

Reflective answer to 2:

The usual computer games are either hand-eye (with occasionally some small amount of brain-) coordination contests, such as "Lunar Lander," or fantasy-land, interactive do-it-yourself storybooks such as "Dungeons and Dragons." These activities take part in the allure of broadcast television: namely, they involve the student kinetically and emotionally, but they do not have a cumulative component. You can walk in on television (or computer games like "PONG" or "SPACE WAR") anytime; no prerequisities or logical-deductive skills are required. (For an excellent exploration of this theme, see Postman, 1979.)

We cannot call this kind of attention passive -- observe any kid watching an action TV show or playing a video game. Nevertheless the interaction is non-analytical. It has more in common with baseball than with reading, more of recess than of curriculum. No wonder teachers of computing have game trouble whenever interactive terminals or micros become available.

The problem this paper explores is the development of an introductory computing curriculum built around a kind of structured gaming. The computing community has begun to understand that carefully chosen programming language features can guide our thought in ways that make code work across time, that is, remain adaptable, comprehensible, repairable. We propose that a similar choice of *gaming* features can foster the development of logical problem-solving skills, while retaining the kinetic/esthetic motivational structure of video games. (We have all known programming hacks who have made the game/program connection.) We want to use microcomputer color graphics to make computing more like color crayons and less like arithmetic.

Having said that much, we will answer Question 1 and then flesh out our answer with a description of the curriculum we are developing.

Question 1: When is a computer game an ok class activity?

Answer:
1) The game must be designed with a set of concepts and skills in mind and a plan for how the game teaches these;
2) The things learned in the activity must contribute toward a *cumulative body of knowledge*, a toolkit that the student can perceive and make use of, as toys and toy-making tools; and
3) The game must be *superseded* by a more interesting, more interactive game, chosen with extreme care to be unplayable unless the student has mastered the skills taught in the previous lesson/game.

CHALLENGE

Our mission, in the University of Tennessee/NSF High School Computer Science Curriculum Project (HSCS), is to make computer skills available to average students. Computers may indeed become as ubiquitous as telephones and televisions, although we believe that the introduction of another technology as soporific, captivating, and anti-thinking as television could be a major social disaster. We hope, rather, that computers will become convivial tools like the telephone; "convivial" means that their use is determined by the user, not

---

This work is partially supported by NSF Grant SED-79-18991

by some central least-common denominator such as broadcaster. We don't have utopian ideas as to what future generations will do with computers. (Who could have predicted in 1915 what problems we'd have with automobiles?) We do, however, have a strong feeling that the question of whether individuals will be able to *program* their computers, or merely *buy* programs, is an open and important question. The challenge, then, is to give every citizen who can dial the telephone some ability to program a computer.

METHOD

This section will be brief; we have published elsewhere (Aiken, Hughes, Moshell 1980) the nuts and bolts description of HSCS. We are using a cartoon-animation software system called RASCAL which runs as part of UCSD PASCAL on the APPLE microcomputer. The basic installation costs about $3200, including a single floppy disk, color television, and 100 character-per-second printer. Each lesson in a one-semester (18 week) course consists of approximately a week of work, divided into these parts:

Introductory activities
Exploration project
Skill-building project
Buttoning-up activities.

A class consists of about 15 students per computer (our collaborating schools have only one APPLE each; we hope to try the curriculum in multi-computer classes later). Five groups of three students alternate computer use with planning work using graph paper and marker pens. The off-line students are planning their strategies, doing hand simulations and observing the on-line students, for to graduate to the next activity, a group must successfully predict the outcome of an assigned "seed" (e.g., geometric pattern, algorithm, program). The activities develop during 18 weeks from a non-linguistic, color-pattern process called "quilting" (next section), through immediate-mode and straight-line-code entry of TURTLEGRAPHICS (Papert,1970) commands and the introduction of PASCAL control structures such as REPEAT... UNTIL and IF...THEN, to the creation of cartoon characters and their animation with complex programs using the RASCAL animation system. The output is always color graphics and music; the curriculum steadily increases its interaction as students learn how to use the joystick to control motion. There is always an underlying lesson about how programs work. All code is in a completely structured language (PASCAL) and is taught from the inside out. Only at later stages do environmental details such as declarations become of concern. A PASCAL interpreter is used which scrolls the source program being executed on the bottom of the screen (at a controllable rate) while the program produces its output on the top part of the screen. A working system will be on exhibit at NECC2.

We will conclude with fairly detailed explanation of the first two lessons, quilting and TURTLEGRAPHICS.

TWO EXAMPLE LESSONS

Lesson 1: Quilting

Behavioral Objectives: Students should learn how to insert a disk and start the system. They will run the Quilt program, explore its features and limits, and use it to generate static and moving color patterns.

Concepts: Students will learn the following:

1) how to enter an initial pattern of information into the system which then controls the repetitive behavior of the computer (*Introduction Activities*).
2) how to construct simple experiments to determine the parameters of the computer's operation, such as how many colors it has, how big the screen is (*Exploration Projects*).
3) how to hand-simulate a formal process of discrete steps to predict the computer's behavior and understand it(*Skill-building Projects*).

Resources:

1) The Quilt program. Students insert a disk and turn on the computer. When the greeting message appears, they type X (execute). The computer asks EXECUTE WHICH FILE? The student types QUILT and a return. QUILT then prints the following message:

```
THIS PROGRAM LETS YOU DRAW A PIC-
TURE AND THEN MAKE A 'QUILT' OF IT
BY REPEATING THE PATTERN. TO DRAW:
SET PADDLE 0 FOR A COLOR;
TYPE KEYS AROUND K TO MOVE THE DOT.
   FOR INSTANCE, 'I' MOVES THE DOT
   UPWARD.
TYPE 'R' TO SEE THE PATTERN
REPEATED.
TYPE 'H' TO HALT THE REPEATING.
TYPE 'C' TO CLEAR THE SCREEN.
TYPE 'Q' TO END THE PROGRAM.
NOW HIT THE RETURN KEY TO BEGIN.
```

When the student hits the return key, the following reminder message remains on the bottom of the screen:

```
REMINDERS:
KEYS AROUND,MOVE:I=UP,J=LEFT,U=DIAG,
ETC
C)LEAR  R)EPEAT  H)ALT THE REPEATING
Q)UIT THE WHOLE PROGRAM
```

The game paddle supplied with the APPLE computer consists of a knob controlling a computer input; the student turns it and observes that the dot in the center of the screen changes color. She types an experimental sequence of keystrokes such as LLLI and observes that the dot moves and leaves a colored trail shaped like Figure 1. She then tries typing R. The computer immediately reproduces the pattern, starting where the cursor was left and filling the screen with a periodic pattern. (Since the screen is wrapped, when the cursor dot falls offscreen left, it returns on the right, and similarly for top-bottom.) If you R)epeat the above seed, you get Figure 2.

2) Miscellaneous resources: custom-made 40x40 graph paper, five or six sets of color crayons.

Sequence of Events:

The first day of this lesson consists of introducing the program and allowing students to play with it for a few minutes each. They are sent away with copies of a sheet of exploration projects to be undertaken the next day. These projects are carefully ordered by increasing difficulty so that the first group on the machine will have some chance of success, while the second group will have to do some planning before getting on the computer (while the first group is working) in order to succeed. The exploration projects are:

Group 1: How many colors are there? Are they distinct, or are some repeated? How are you going to be sure?

Group 2: Can you paint with black, or is it like the paintbrush not touching the painting? How high and wide is the screen? How many colored blocks are there?

Group 3: How many blocks (pixels) per second can this computer draw?

Group 4: Can you find a pattern (seed) which, when run, will fill the entire screen with one color? What is the shortest one you can find?

Group 5: Let's say you were allowed only seven keystrokes to make your seed. If I stuck a piece of tape on the screen somewhere, can you make a seed which will zap a line through my marked spot? (It might go around several times; that's ok.)

The second and part of the third days are spent giving each work group (two or three students) ten minutes access time to the computer, while the other groups watch or plan. The teacher moves about observing, providing hints, and showing students how to pretend to be the computers using graph paper and crayons. We turn the scarcity of computer access to our advantage here by requiring the students to be careful and precise in their hand-simulation of the repeat-process in order to predict what their programs will do.

During the third day the teacher determines which groups have completed their tasks and gives them the next challenge sheet: the skill-building projects. Since the first groups got easy exploration projects (and finished earlier), they are given tougher skill-building projects. Groups that haven't figured out their explorations by the middle of the third day are helped to find the answer and moved on.

The object of skill-building exercises is to be able to accurately predict what a given seed will do. Two parts are given. In part I, each group is given a different list of three or four seeds and asked to hand-simulate the result and then to try it. When they can do this assignment correctly, they move to part II. Here, the worksheet gives them the finished pattern, and they are to find the seed that creates it. In this exercise, the game really becomes like a game. Moving patterns are possible, for instance, if the seed goes back over itself in the background color before continuing. Again, because the students are having to hand-simulate during their off-machine time, skills are being built while students are planning success. The teacher might only give very short machine access times during this phase, requiring that a filled-out crayon simulation be presented as a ticket to allow its being tried on the computer.

On the fifth day the class is given over to a show-and-tell, in which each group explains how it found the answer to its exploration question and shows off the most interesting seed. Polaroid photos or slides of each group's best pattern are taken for use in the end-of-course Parents' Day exhibit.

Lesson 2: TURTLEGRAPHICS

Behavioral Objectives: Students will learn how to control the screen turtle via individual commands to the INTERP program. They will explore the features of TURTLEGRAPHICS and of NOTE, a music-output function.

Concepts: Students will learn the following:

1) the idea of using commands that are words to get desired output. They should understand that commands consist of operators (verbs) and operands (nouns or adjectives).
2) the concepts of experiment construction and hand-simulation (which are introduced by Lesson 1 and reinforced here).

Resources:

1) INTERP, the PASCAL interpreter program. INTERP will be used throughout the course. In this lesson, the immediate mode will be used; as each command is entered it is executed. The commands used this week (listed below in Day 1 Events) control the position of an imaginary screen turtle which leaves a colored trail as it moves. A music-making command is also explored.
2) Six small protractors.
3) Six pre-made, but unmarked, cardboard rulers long enough to reach diagonally across the TV screen.

Sequence of Events:

On the first day, the teacher shows students how to start INTERP and enter the immediate mode, then types the following six lines, pausing after each while the class observes the effect:

```
FILLSCREEN(BLUE)
FILLSCREEN(BLACK)
PENCOLOR(WHITE)
MOVE(20)
TURN(50)
MOVE(20)
```

These commands are written on the blackboard. One more command is added, MOVETO(20,20), to be used if the line being drawn goes off the screen. Students are allowed two minutes each on the computer to try these commands. The concept of commands (operator, operands) is briefly explained.

Four discovery questions are then given to the class:

1) What is the size of the screen? Where is the center?
2) How many angles are there? What do negative angles do?
3) How many colors are there? What are they? What happens when various pencolors and background colors are used together?
4) Try the command NOTE(number, number). What does it do? What is the effect changing the first number? The second number? What are the legal ranges of each number?

Each group is given responsibility for one question. Two groups get question 4.

On the second day, work groups attempt to answer their questions. They are given a description of the concept of shooting a dot, which they may try when they have answered their questions. (They need those answers to shoot the dot.)

To shoot a dot, a piece of colored tape is stuck to the television. The students try to pass a turtle track through the dot. Doing it with one TURN and one MOVE is a successful shot. The teacher will provide protractors and (blank) rulers and will suggest that students make television rulers, but without telling them how.

After having demonstrated the ability to shoot a fixed dot on the screen (beginning of Day 3), the team will graduate to production geometric figures of the following types -- increasing in difficulty:

1) square
2) rectangle
3) isosceles right triangle
4) equilateral triangle
5) rectangle with diagonals
6) trapezoid
7) 5-point star
8) Star of David
9) "naked dandelion" (radial lines from a common point)
10) a short name, in block letters

The first five designs above will be attempted by the teams on Day 3, with the first team to the terminal getting design 1, the second team getting design 2 and working first at their desks with graph paper, protractors and rulers, and so on through all the teams. On Day 4 or after they have mastered all of the first five designs, the teams will tackle designs 6-10. On Day 5, the class as individuals will be given TURTLEGRAPHICS commands to produce a design at about level 8 above and asked to draw the figure on graph paper as it would appear when drawn by TURTLEGRAPHICS.

THE COGNITIVE STYLE OF THE HSCS CURRICULUM

At this point the reader may ask, "Why do you call these activities games at all? What have your lessons in common with Blackjack or Lunar Lander?"

The fact that unifies Quilting, TURTLEGRAPHICS, and more traditional interactive games is that they sell themselves. No one has to compel students to do their assignments.

The point at which our curriculum diverges from closed games is that the only real opponent in traditional games is a pseudo-random number generator or perhaps another human. In a cognitive game the opponent is the rich structure of our own ignorance. The excitement of being able to create pattern and order is as old as the wall paintings in the caves of France. It is an essentially human activity, one at which all players can win. It is also a meta-game, in which an infinite number of specific games like shoot-the-dot can be expressed. The relationship between cognitive games and traditional games is analogous to that between a set of blocks and a preassembled toy. A different order of learning becomes possible.

THE DESIGN STYLE OF THE HSCS CURRICULUM

The fundamental design principle we have followed is to attempt to make each lesson augment the student's skills in three areas: discovery, control, and design. We allow students to play with the system as each new feature is introduced, but they have discovery questions whose answers they seek as they mess around in more or less structured ways. They need to find the answers to be allowed access to the next level of the system. Students develop discovery skills by experimentally answering questions like "what does this command do?"

We ask students to undertake specific challenge, such as the shoot-the-dot game, to develop their ability to control the computer by selecting the correct command and providing correct values for its operands. Their understanding of the system is built by simulation exercises, which allow them to predict the behavior of a command and thus to choose the right command.

Later in the semester, students will begin writing programs; but even at early stages there is the impetus to design input sequences to produce the desired pattern. Students must be able to produce a sequence of commands which produces the predicted output on first submission in order to graduate to the next level of the system.

Another principle we have followed can be summed up in the phrase "design from the first experience." We believe that computer science (or anything else) should be taught from the inside out. That is, first experiences must incorporate the heart of the matter at hand, with as little extraneous matter as possible. For instance, Quilting teaches the fundamental core of the computing experience: in repetition of a controlled process there is great power. The Quilting lesson is taught without introducing a word of jargon, previous assignments, or complex command sequences. Quilting, and its fundamental message, can be taught to illiterates. The second lesson similarly teaches the relationship between operands, operators, and results. Only after students have firm operational skill with a given tool, do we introduce terminology, written reference materials, and the ultimately necessary environmental details such as data declarations and control statements.

This paper has addressed the important question of "Quo Vadis" microcomputer education? We are excited by the prospect of transforming gaming, a traditional problem area for computing teachers, into one of their primary tools.

The authors acknowledge and appreciate the assistance of their collaborators: R. M. Aiken, C. E. Hughes, C. R. Gregory and J. A. Ross (University of Tennessee); L. Demarotta (H. C. Maynard High School); E. Miner (Alcoa High School).

REFERENCES

Aiken, R. M., Hughes, C. E., and Moshell, J. M., "Computer Science Curriculum for High School Students," Proceedings ACM/SIGCSE Conference, Kansas City, Montana, February 25, 1980.

Papert, S., "Teaching Children Thinking," Proceedings IFIP World Congress on Computers and Education, Amswerdam, 1970.

Postman, Neil, "The First Curriculum: Comparing School and Television," Phi Delta Kappan, 61:3, November 1979.

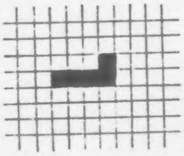


Figure 1: The Seed Pattern

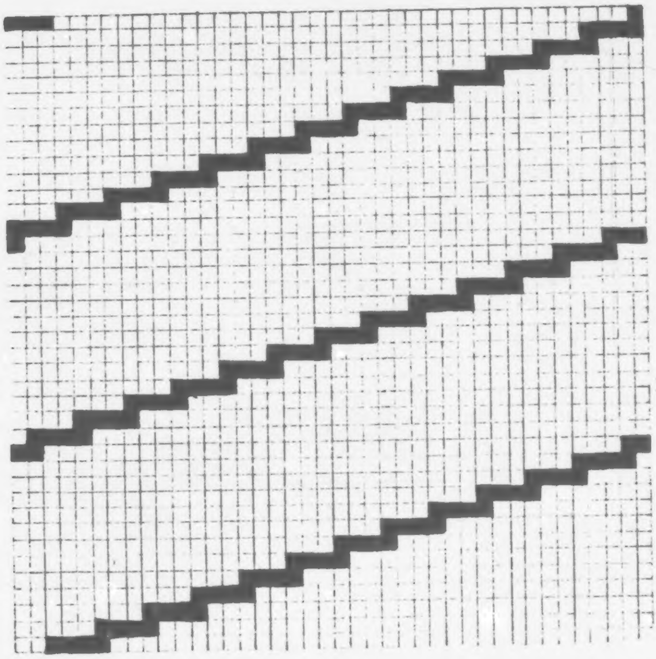


Figure 2: The Repeated Pattern

TAPPING THE APPEAL OF GAMES IN INSTRUCTION

Peter O. McVay
Educational Services
Digital Equipment Corporation
12 Crosby Drive
Bedford, MA 01730
Tel. 617-275-5000 Ext. 2217

Games and the Computer

Around the time when computers began appearing in classrooms and ceased being a novelty, the theory arose that the ideal way of teaching with a computer was to put instruction in the form of a game. Actually, this teaching method has been around for years; pioneers in the Dewey teaching method used many games in their curriculum, and a number of Montessori techniques also involve games.

Probably one reason the game idea so caught and held the attention of educators with computers was the mania for computer games in general. Really brilliant students (and instructors) spent an inordinate amount of time designing and programming new games. Other instructors and students spent a huge amount of time playing these games. Why fight it? Design instruction to take advantage of the built-in motivation; there is a pool of course developers anxious to develop instruction (i.e., games) and a large audience of enthusiastic students (players).

What Happened: Expectations Denied

With a few exceptions, the results were notably disappointing. Problems arose on two fronts when games were made an integral part of computer-aided instruction:

> The game quickly took over the instruction and frequently became more important than the content of the lesson.
>
> Language-oriented subjects required a huge amount of programming effort to bend to an effective computer game. Human languages and thought processes simply are not easily transferred to a computer, except on the simplest level. [Note the word "easily"--there are brilliant exceptions to the above statement.]

The pendulum recently has appeared to swing in the opposite direction: games are now anathema and viewed as a frivolous pastime at best. But this approach ignores that games are immensely popular and have a tremendous attraction for game players and designers both. This paper proposes to extract some of the good points about games and then apply them to real computer-aided instruction.

Salvaging Valuable Parts

What points are worth saving? What can games do that other methods of instruction do not do as well? The observable characteristics of computer games (and computer game players) are:

> A high level of motivation. Game players and designers spend hours working at the terminal.
>
> Clear and consistent goals. All true games have a clear ending in mind. How much instruction becomes bogged down because of muddy goals and objectives?
>
> A high amount of player interaction. Game players are doing something: they are manipulating the terminal, the computer, and the game structure itself.

Maximum choices for the player. Studies of children's toys have shown that the flashiest toy is not the one that is played with most often. The toy that gets the most attention is the simplest one that allows the child to use imagination. The most popular games (on or off the computer) are those with the simplest constructs and widest range of choices.

Simplicity. How much instruction fails because the starting sequence for the student is too complex? Complexity in itself is not the horror--the problem is that no one bothers to explain all the details to the user. Most good games come with very detailed instructions. [Note: why does the programmer who balks at providing documentation churn out reams of instructions for the game he just designed?]

Creativity. Many of the aspects of computer games--whether they are traditional or invented by the user--have highly creative parts. An axiom among game designers that is frequently used to justify their interest in games is that some of the most highly creative ideas and programs are first developed in a game.

Applied Gaming Principles

How can the principles from games be applied to instruction? It turns out that the items that make games so appealing are intertwined, and by incorporating several facets of games into genuine instruction, several objectives can be achieved at once (e.g., high motivation, strong interest, good interaction).

The remainder of this paper is a checklist of items to look for in any computer-aided instruction. These are items which have been found to most reliably increase the quality of computer instruction and, it is hoped, the skills of students taking the course. An important point about this list is that it is not meant to be exhaustive--it is a start. Persons wanting to use the questions as a checklist may also have their own principles and procedures to add to the list.

Questions for the Developer

Is the method of presentation consistent with the material? The best computer games require player actions that fit the total concept of the game. Players move tokens by giving directions to the computer in a manner similar to picking up the piece by hand (for board games). If the game is a thought-type game, the player makes decisions that are consistent with real-world decision methods. The computer in both instances is a referee, informing the player of the consequences of his actions.

Poorly designed instructional games or simulations use methods of advancing players (students) that are not related to real actions. In a number of computer board games, students race cars, horses, or other items around a track. But instructional racing games make the pieces move around the track by answering questions. These games are not notably successful because the action is too slow and because races are not normally run by answering questions. This design error occurs because the developer has inferred that actions that game players enjoy in one setting must be good in all settings. This mistake results in a product that is neither good instruction nor a good game. Lesson developers can avoid this trap by chosing a presentation method without regard to the observed popular appeal of a method in a different setting. A natural, or consistent, presentation enhances the appeal of a lesson considerably. If the subject is math, the game should use mathematics naturally, not as a means of moving a token. The Minnesota Educational Computer Consortium has developed economic models that allow children to run simulated businesses (lemonade stands, bicycle factories); both mathematics and economic fundamentals are taught through these games. English teachers can use word processors available on most computers in their classes. Students take naturally to word processors--they enjoy seeing their work turned out in a professional format. There are many other examples--the key is to ensure that the presentation matches the material.

Is there a large amount of player interaction? Frequently the charge raised against television can also be raised against education: the student sits and absorbs the material passively. The lecture format certainly has its place, and brilliant and stimulating lecturers are rare individuals that should be sought after. But computer-aided education as a lecture or electronic page-turning format usually does not rise to these heights--and is also a bad use of the medium.

An examination of the most popular games shows a high amount of player interaction. This interaction is not simply key-pushing; some of the popular games are also limited to single-key responses after lengthy actions by the computer (football, adventure, road-race). But the player is constantly thinking--decisions must be made, paths chosen, strategies worked out.

Again, an objection will be raised that the material in education cannot be adapted to such a lengthy decision-making format. But an examination of the actions that students take when studying shows several possibilities for increasing the student's participation:

1. Path choice. When a student is studying away from the classroom, he at least has the option of choosing which page or chapter to start on. Why not include this choice in the computer session?
2. Notes. It is relatively simple to provide an electronic scratch pad--each student can be provided with a comment or note option during the lesson. At the end of the lesson, the student can either store the notes or (if disk space is limited) take a printout of the notes away.
3. More inquiry. This does not mean more questions. Inquiry can be a form of path-taking, or an invitation to speculation. The student chooses which subject to take first. Incidentally, this also answers the frequent question of the correct order to study subtopics. The student can choose the topic and also skip from topic to topic if necessary. The same material is eventually covered, but along paths that the student has chosen.

Is the material creative? One puzzling problem in computer-aided instruction is that the same individuals that produce ho-hum instruction also are turning out brilliantly designed games. Obviously the same effort is not going into both projects. This problem is a management, not a computer, problem. There can be several reasons for this dichotomy when it appears:

1. Freedom to experiment and make errors. If you insist on error-free and perfect instruction from your developers, that is exactly what you'll get--with all the creativity and originality of the Saturday-morning cartoon shows. A perfect lesson is not necessarily a good lesson. In fact, lessons that have incomplete sections may even be more useful, since the student must fill in the missing portions.
2. Appropriate comments at appropriate times. Kibitzers seem to abound in educational development areas. The discussion of approaches and critical analysis of lessons under development is an important part of the creative process, but there are many cases where a promising bit of instruction has been discarded because constructive criticism was heaped upon the material--before it was developed and before advice was requested. While a developer is writing a game as a hobby, no one is leaning over his shoulder. Leave the developers alone until there is something to criticize other than a first draft.
3. Responsibility. Managers frequently lack confidence in the persons working under them, and hope to avoid errors by giving very detailed instructions to the developers. If a developer is presented with a cut-and-dried package, there is little room for a new and creative approach. For someone to become truly involved with a project, she must feel that the project is at least partly hers, and must have some involvement with planning. And responsibility extends down to some very low levels. In a language arts project in Norfolk, Virginia, the computer operators were employed to enter some of the questions students would be given. They were asked to use their imagin-

ations when entering the praise response students would receive on correct answers. The results were a series of questions with highly original responses that the students could relate to. The operators also took a personal interest in the development of the course, and frequently monitored student progress and asked to change lessons that were not popular.

Is everyone using the material enthusiastic? Here is another factor that is a function of the people involved and not the computer. "You WILL be enthusiastic!" is obviously an unworkable approach (but one which is sometimes tried!). But if everyone involved with the development of a particular piece of instruction finds it tedious and difficult to work with, then perhaps the entire instruction set should be looked at. If persons who are working in their subject area find the topic boring, then what about the students? What is needed is either a fresh and creative approach (see above) or a different topic. If a subject is completely boring, then its value is open to question. Tepid topics usually result if the individuals can see no value in them whatsoever.

Enthusiasm, once engendered, tends to be catching. Enthusiasm most frequently stems from freedom and responsibility--two subjects discussed under creativity and maximum choices, above. Freedom and responsibility extend to all levels: supervisors, developers, teachers, and students. Teachers and administrators tend to be wary of allowing students to take part in the teaching process, and for good reason: student freedom is difficult to manage and control. Real participation in their own education is also a novel idea for most students; they have been passive consumers of instruction for so long that they may not be able to handle such a responsibility without careful pre-training. But the results can be spectacular.

Here are some areas in the instructional process where personnel at all levels can directly participate:

1. Level of instruction. Too hard? Too easy? Should it be presented to a different grade level or in a different subject area?
2. Method of presentation. Is the format (question and answer, screen display, essay and choice) appropriate for the subject?
3. Effectiveness. Does the content of the lesson stay with the student?
4. Correctness. Is the content free of grammar, syntax, spelling and content errors? (Here is an area where younger students delight in showing off. If you choose to release them on this one, prepare for an avalanche.)
5. Additional lessons. What other material should be added or changed?
6. Design of instruction itself. Developers can work with older students, or students who have just completed the course, to redesign and evaluate the instruction.

Is the course simple and direct? One fault of some of the innovative courses is that they tend to be immensely complex. While at some time in the future humans may learn to think recursively and in complex algorithmic patterns analogous to the machines they use, presently people still think in the same old way. Presenting highly complex study structures, maps, objective fields, and learning paths can set up a forest that quickly discourages even the most determined students.

If the structure is too complex, there are three solutions:

1. Check the presentation. In complex structures, a simple, direct path can frequently be found in the material.
2. Turn routing over to the student. If given a list of topics and objectives, the students can make their own choices of what to study first, and when.
3. Hide some of the complexity. Some of the structure may be due to the enthusiasm of the developers for marvelous structure. A lot of the design can be safely hidden by having the computer do the routing work.

Great advances have been made in the use of computers in classrooms in the past few years, and the pace of development and discovery increases with each new application. The procedures discussed in this paper are a starting point for developing better computer-aided instruction. The key to successful computer-aided instruction, as in any other form of instruction, is the people involved in the project. There is no substitute for a creative, dynamic, and highly qualified individual in the right position at the right time. Fortunately, these persons appear to be available because computer-aided instruction has made great gains in recent years. The continuing analysis of what constitutes good technique in the use of computers will continue this trend.

Computing Curricula

AN EDUCATIONAL PROGRAM IN MEDICAL COMPUTING FOR CLINICIANS AND HEALTH SCIENTISTS

Albert Hybl
Department of Biophysics
(301) 528-7940

James A. Reggia
Department of Neurology &
Department of Computer Science
(301) 528-6484

UNIVERSITY OF MARYLAND
School of Medicine
Baltimore, Maryland 21201

ABSTRACT

The growing importance of computers in medicine implies that health professionals lacking rudimentary knowledge of their potential use will be handicapped in the future. In this paper we discuss the content and implementation of a curriculum in medical computing that we have introduced at our medical school to remedy this problem. In addition to formal classroom instruction our program includes such innovative features as a demonstration laboratory for exhibiting computer applications in medicine, the use of a family of knowledge management languages that are designed for computer-inexperienced clinicians, and the involvement of interested individuals in ongoing research in computer applications in medicine. Our description of this curriculum and its implementation should be of interest to anyone involved in teaching computer-inexperienced individuals about the potential uses of computers.

INTRODUCTION

In recent years the role of computers in medical education has been receiving a great deal of attention. The vast majority of this interest has centered on computer-aided instruction for a wide range of clinical and pre-clinical topics (3, 4, 8, 9, 14). At the present time many medical schools use such automated instructional material to supplement conventional teaching methods.

In this article, however, we will discuss a complementary aspect of computers in medical education: introducing medical students, practicing physicians, and other health-related personnel to the applications of computers in medicine. Educating individuals in the health professions about medical computing has recently been singled out as an area of great importance that deserves more attention than it has received (7, 12, 18). Very little literature has addressed this goal and there has been no clear consensus about which aspects of computer science should be taught to medical students and physicians. How this material should be taught and how it should be introduced into an already information-dense medical education process have been given little if any attention.

It is our belief that the growing influence of computers on medical care makes some familiarity with their uses and potential important to health professionals. We have thus designed and implemented a program to supplement conventional medical education with instruction on medical computing. This program has two major objectives:

(1) to provide health professionals with a broad understanding of current and potential uses of computers in medical research and clinical practice; and

(2) to give health professionals the ability to directly use available computer resources for solving clinical and research problems.

In this report we begin by outlining the content of a curriculum on computer applications in biomedicine that we believe achieves these objectives. We then discuss how we have approached the implementation of this program at our institution and the response it has generated among medical students and faculty members. Finally, our experiences are reviewed in the context of other recent endeavors in this area.

A MEDICAL COMPUTING PROGRAM

The content of our medical computing program reflects our feelings about what topics are currently important or will become so in the future. Our choice of topics is based largely on medical computing systems that are presently used and a review of recent literature on biomedical computing research. Our emphasis is on the use of the computer as a tool to accomplish various tasks rather than the fundamental concepts of computer science. The material in our program can conveniently be divided into five sections:

- General Concepts
- Scientific and Laboratory Applications
- Clinical Applications
- Educational Applications
- Administrative Applications

The section on general concepts is tailored to increase student awareness of the capabilities of computers and to provide them with the fundamental knowledge needed to use a computer. It includes:

- Essentials of Interactive Computing: In order to encourage students to become computer users, we begin by explaining the mysteries of account numbers and passwords, the techniques for signing on and off, the idiosyncrasies of using certain terminals, the marvels of the executive system, and how to save data and/or programs in retrievable files. (Batch processing of programs is discussed but not used by students.)

- Document Processing and Text Editing: These systems are easy to learn and immediately useful to the students. Their mastery facilitates the learning and use of other computer resources.

- General Information Processing: Programming languages for information processing and for general problem-solving are surveyed. Guidelines are presented for selecting an appropriate language for various tasks.

- An Overview of Applications: An initial overview of the range of scientific, laboratory, clinical, and educational applications of computers in medicine is given. This includes discussion of local working systems such as a whole-body CAT scanner; a cranial CAT scanner; two nuclear scanners; computerized basic science research laboratories; the QS-2 system for ICU patient data collection, storage, and monitoring (at the Maryland Institute for Emergency Medical Services); a partially implemented PROMIS system (17) (at the Baltimore Cancer Research Center); and the Stroke Database Center (located adjacent to the Neurology Ward, this unit serves as an entry point to the multicenter National Stroke Database (11)).

- Social and Legal Issues: A discussion of the present and future effects of computers on the practice of medicine is given and includes material on ethical issues, standardization, and the economics of medical computing. The Privacy Act of 1974 (PL 93-579), the Federal Brooks Bill (PL 89-306), and federal information processing standards are examined.

The section on scientific and laboratory applications contains the following material:

- Real-Time Instrumentation: Configurations of microprocessors and minicomputers for data acquisition ranging from stand-alone dedicated facilities to hierarchical computer networks are discussed. Techniques for collecting and communicating data to the computer are stressed.

- Graphics: The use of terminals with graphics capabilities and plotting systems is explored. Their ability to extend the effective use of computer resources by improving the presentation of certain types of information is covered. The plotting of results from the digital simulation of a biophysical system is used as an example of the supportive role of graphics.

- Image Processing: The role of computers in various clinical scanning systems (e.g., CAT, nuclear) is discussed. Attention is given to the future potential of image analysis in the laboratory (e.g., chromosome analysis, cell counting).

- Statistical Analysis of Data: The basic concepts of data analysis are introduced and available packages for statistical processing of data are described (e.g., SPSS, BMDP). Example applications such as epidemiologic data analysis are provided.

The section on clinical applications covers topics dealing with the use of computers in patient care and clinical research including:

- The User-Computer Interface: The use of computers for obtaining a patient's history and the problems of making computer facilities directly usable by physicians are explored. A brief introduction to natural language processing by machine is provided.

- Medical Information Systems: This includes the use of registries, data bases, and hospital information systems for storing and retrieving medical records for use in patient care and clinical research. Aspects of security and integrity are examined in detail. The concept of a query language is introduced and currently available systems are reviewed (e.g., COSTAR, PROMIS, ARAMIS).

- Computer-Assisted Medical Decision-Making: The potential uses of computers for assisting physicians with diagnosis and patient management are discussed. Concepts of knowledge representation and alternative inference methods (e.g., statistical pattern classification, production rule systems, cognitive models) are covered.

The section on educational applications covers material relating to any aspect of the health professional's training. It includes:

- Computer-Aided Instruction: The available systems for computer-aided instruction (e.g., PLATO, ASET, HEN) are reviewed and information on how to author suitable lessons is presented.
- Education in Medical Computing: The attitudes of health professionals toward using the computer for medical applications are explored. The concept of a curriculum on medical computing and a survey of existing or proposed programs are presented.
- Reference Systems: Computerized literature searches (e.g., MEDLINE) and more advanced medical knowledge bases are described.

The section on administrative applications surveys the use of computers in hospitals and physician offices for scheduling, accounting, analysis of policy-related issues, and other related tasks.

CURRICULUM IMPLEMENTATION

Currently we have implemented formal coursework that encompasses most of the material outlined above. This coursework began with only a single one-credit minimester course that covered essentially all of the material listed above at a fairly superficial level. We have recently added a second one-credit minimester course and divided the material between the two courses permitting it to be presented in greater depth. Although both courses begin by familiarizing students with the essential elements of interactive computing, they are largely complementary in what they cover. One course addresses general concepts, scientific and laboratory applications, educational applications, and administrative applications. The other course concentrates on clinical applications. Both courses are offered twice a year and are taught in a coordinated fashion. Students are permitted and encouraged to take both courses.

Since biomedical computing has not been generally recognized as part of the medical curriculum, our approach has been to gradually introduce this material into the medical school program. We feel that the development and evolution of our program has profited from an incremental implementation. The gradually increasing amount of coursework appropriately parallels the amount of computer use by the medical community.

To maximize the student's experience, we have adopted a teaching approach that uses several different instructional methods. Much of the material is presented as formal lectures that are complemented by related reading assignments from the current literature. The use of lectures and reading assignments allows us to cover the wide range of concepts outlined above in the brief time available. However, to give medical students a more intimate acquaintance with the impact of the computer on health care and medical research, we have supplemented the traditional teaching approach in three ways.

First, students are made cognizant of the diversity of applications through tours to on-campus facilities where dedicated computers are in active use (the sites are listed above under "An Overview of Applications"). These visits effectively emphasize the present role of computers in the health professions and contribute to student motivation to learn more about biomedical computing.

The second way we are supplementing classroom teaching is through the development of a laboratory for the demonstration of applications of computers to specific research and to clinical and educational problems that are not currently in day-to-day use at our hospital. EKG analysis (currently being implemented) is especially suitable for demonstrating the major components of laboratory computing: data acquisition (analog-digital signal processing), feature extraction (R-wave recognition, QRST classification, power spectrum analysis), and decision-making (presence or absence of myocardial infarction). Several computer-aided decision-making systems are complete (e.g., diagnosis of thyroid disorders and stroke; prediction of outcome following cardiac arrest) and are used to illustrate the current level of state-of-the-art medical decision support systems. Feedback from medical students and physicians is useful for guiding the development of the experimental systems that are being demonstrated.

The third aspect of our instructional approach involves hands-on experience using the computer facilities at our university. In one of the minimester courses students are expected to use the Document Processing System to prepare a critique of a current journal article of their choice. The students are also given projects involving digital simulation, computer graphics, general problem solving, and using statistical packages. In the other course that covers mainly clinical applications, students are asked to develop two small computer-aided decision-making systems using a family of knowledge management languages designed for use by computer-inexperienced individuals (15). The systems supporting these languages provide a complete computer-aided decision-making system when given a description of the problem to be solved and the relevant knowledge needed to solve it. By using the computer for these projects students are able to learn directly about the capabilities and limitations of the computer for assisting them with a wide range of tasks.

At the present time our elective minimester courses are reaching about 5 - 10% of the freshman and sophomore classes. Students taking these courses have a wide range of computing backgrounds, varying from essentially no previous computer science to an occasional student with an undergraduate degree in a related field (e.g. biomedical engineering). The variety of instructional approaches used has been fairly successful in making the material presented understandable to computer-inexperienced students. At the same time the broad scope of the material has provided a challenge to even the most advanced students. Student evaluations of the courses implemented so far have been essentially positive; other faculty members and university administrators have begun to take an interest in the program.

DISCUSSION

Previously proposed or implemented programs for education in medical computing fall primarily into two categories. First, and most common, are those representing complete curricula that consist of interdisciplinary studies in medicine and computer science. These curricula are designed to produce individuals who are specialists in biomedical computing and thus typically lead to graduate degrees. Several examples of such curricula have been described (1, 6, 13, 19) and a recent review of relevant programs in the Federal Republic of Germany has addressed this issue (10).

The second category of educational programs in medical computing include those consisting of only a single course designed to supplement the education of health professionals (e.g., nurses, pharmacists). For the most part, these courses have covered a very limited range of topics (2, 5, 16).

The medical computing curriculum that we have described in this report lies in between the two different categories outlined above. On the one hand, it is designed to supplement the traditional educational experience of medical students and other health professionals similiar to the single course programs. On the other hand, it consists of more than a single course and covers a wider range of topics. The depth to which these topics are covered is gradually increasing as the available coursework increases.

In this report we have attempted to describe not only what the content of a medical computing curriculum should be, but also how it might be implemented within the constraints of the traditional medical school experience. In particular, we have presented our belief in the need for an incremental implementation and an argument for supplementing conventional lecture material with hands-on computer experience, visits to relevant facilities, and the use of a demonstration laboratory.

Our present plan is to continue the evolution of our minimester elective program through additional courses and by instituting a more formal evaluation of their success. The demonstration laboratory needs further work; we need to develop and acquire additional software and hardware. Long-range goals include the establishment of medical student externships in advanced topics in medical computing, integration of at least some form of required instruction on biomedical computer applications into the standard medical curriculum, and the creation of a continuing medical education course to reach practicing physicians.

Acknowledgements: We thank the Departments of Diagnostic Radiology, Pharmacology and Experimental Therapeutics, and Radiation Therapy, the Maryland Institute for Emergency Medical Services, and the Stroke Databank for graciously accommodating our class tours of their facilities. Computer time for this project was supported in full by the Computer Science Center of the University of Maryland. Dr. Reggia gratefully acknowledges support for this work from an NIH Teacher-Investigator Development Award (5 K07 NS 00348).

REFERENCES

1. Ackerman, L. V. & Harris, D. K. (1975). "Architecture for a Graduate Level Educational Program in the Area of Computer Systems in Medicine," AFIPS Proceedings: 1975 National Computer Conference, 44: 765-768.

2. Buckwell, L. J. (1979). "Education of Health Care Students to Accept and Use the Computer," Proceedings of the 3rd. Symposium on Computer Application in Medical Care, edited by Dunn, R. A., 206-209. New York: IEEE.

3. Burch, J. G, Heyman, A., Hammond, W. E. & Haynes, C. (1978). "Computer-Aided Instruction in Clinical Neurology," Journal of Medical Education, 53(8): 693.

4. Deland, E. C. (1978). Information Technology in Health Science Education, New York: Plenum Press.

5. Dumas, B. P. (1979). "A Computing Course for Pharmacists," Proceedings of the 3rd. Symposium on Computer Application in Medical Care, edited by Dunn, R. A., 210-213. New York: IEEE.

6. Duncan, K. A., Austing, R. H, Katz, S., Pengov, R. E. & Wasserman, A. I. (1978). "Health Computing: Curriculum for an Emerging Profession, Report of the ACM Curriculum Committee on Health Computing Education," Proceedings of the ACM 1978 National Conference, 277-285. New York: ACM.

7. Duncan, K. A. (1979). "Educational Programs for Health Care Computing," Proceedings of the 3rd. Symposium on Computer Application in Medical Care, edited by Dunn, R. A., 204-205. New York: IEEE.

8. Halverson, J. D. & Ballinger, W. F. (1978). "Computer-Assisted Instruction in Surgery," Surgery, 83(6): 633-638.

9. Kenny, G. N. & Schmulian, C. (1979). "Computer-Assisted Learning in the Teaching of Anaesthesia," Anaesthesia, 34(2): 159-162.

10. Koeppe, P. (1977). "Education in Medical Informatics in the Federal Republic of Germany," Methods of Information in Medicine, 16(3): 160-167.

11. Kunitz, S. C., Havekost, C. L. & Gross, C. R. (1979). "Pilot Data Bank Networks for Neurological Disorders," Proceedings of the 3rd. Symposium on Computer Application in Medical Care, edited by Dunn, R. A., 793-797. New York: IEEE.

12. Levy, A. H. (1977). "Is Informatics a Basic Medical Science?" Proceedings of MEDINFO 77, edited by Shires, D. B. & Wolf, H., 979-981. New York: North Holland.

13. Levy, A. H. & Chen, T. T. (1977). "Plans for a Program in Medical Information Science," AFIPS Proceedings: 1977 National Computer Conference, 46: 321-325.

14. Madsen, B. W. & Bell, R. C. (1977). "The Development of a Computer-Assisted Instruction and Assessment System in Pharmacology," Medical Education, 11(1): 12-20.

15. Reggia, J. A. (1980). "A Domain-Independent System for Developing Knowledge Bases," Proceedings of the 3rd. National Conference of the Canadian Society for Computational Studies of Intelligence, Victoria, B.C., Canada. (in press).

16. Rowley, B. A. (1979). "Computer Medicine Educational Program for Medical Students and Others," Proceedings of the 3rd. Symposium on Computer Application in Medical Care, edited by Dunn, R. A., 214-216. New York: IEEE.

17. Schultz, J. R. & Davis, L. (1979). "The Technology of PROMIS," IEEE Proceedings, 67(9): 1237-1244.

18. Shannon, R. H. & Duncan, K. A. (1978). "Why a Curriculum in Health Computing?" Proceedings of the ACM 1978 National Conference, 273-276. New York: ACM.

19. Wasserman, A. I. (1979). "Educational Programs in Medical Computing - Their Basis and Implementation," Proceedings of the 3rd. Symposium on Computer Application in Medical Care, edited by Dunn, R. A., 217-222. New York: IEEE.

A SECONDARY LEVEL
CURRICULUM IN
SYSTEM DYNAMICS

Nancy Roberts
Lesley College
Graduate School of Education
9 Mellen Street
Cambridge MA 02138
617-547-8844

Ralph M. Deal
Chemistry Department
Kalamazoo College
Kalamazoo MI 49007
616-383-8452

SYSTEMS THINKING IN EDUCATION

For quite some time, the influence of science on society and education has been primarily reductionist, seeking to understand phenomena by detailed study of smaller and smaller parts. Overemphasis in this direction has led to a separation of allied sciences like physics and chemistry. "Learning more and more about less and less" (Odum, p.9), as one critic described reductionism, has taken place with almost the complete absence of a movement in the opposite direction to integrate knowledge. Education tends to follow the pattern of parts within parts. Knowledge is divided into disciplines, and disciplines are further divided into subjects. The student is seldom exposed to materials which seek to integrate the various disciplines and subjects.

Education as well as the everyday world of social and economic analysis has a pressing need for synthesis and holism. The systems approach seeks to understand how parts fit together to form a whole that functions for a common purpose. It is in the world around us that most students will be making decisions for the rest of their lives. Explicitly understanding the nature of these systems from a structural and a behavioral point of view constitutes a most relevant and useful education.

THE APPROACH: SYSTEM DYNAMICS

System dynamics is a method for understanding and managing complex social systems. It is built on traditional management (information, experience, and judgment) and feedback theory or cybernetics (principles of structure and selection of information). Mathematical models of varying degrees of complexity are built to reflect the system out of which a problem grows. Computer simulation is used because behavior implicit in the structure of complex systems is too involved to be solved by direct mathematical methods. Model building and computer simulation is therefore an integral part of the system dynamics problem-solving approach.

System dynamics starts from the practical world of observation and experience. It does not begin with abstract theory nor is it restricted to the limited information available in numerical form. Instead it uses the descriptive knowledge of the operating arena about structure, along with available experience about decision making. Such inputs are augmented where possible by written description, theory, and numerical data. Feedback principles are used to select and filter the information that gives the structure and numerical values.

System dynamics was launched in 1961 when Professor Jay W. Forrester of MIT published his book Industrial Dynamics. As the field of system dynamics broadened in its applications, it also broadened in its student audience. Initially taught as a graduate level course at MIT, it has spread to the undergraduate curriculum in many universities. In 1975, as a dissertation project, Roberts introduced the basic concepts to fifth and sixth grade students (1978). The evidence that fifth and sixth grade students could understand and apply these basic concepts led to the development of the current secondary level curriculum project being funded by the Office of Education. The project is based at Lesley

College Graduate School of Education, Cambridge, Massachusetts, under the direction of Nancy Roberts.

A SECONDARY LEVEL CURRICULUM IN SYSTEM DYNAMICS

The current curriculum development group is writing and pilot testing a set of six self-teaching, introductory learning packages. This material will make it possible for a teacher in any discipline to introduce model building and computer simulation as a problem-solving method.

The titles of the six learning packages are:

I. Basic Concepts: Dynamic Problems, Systems and Models
II. Structure of Feedback Systems
III. Graphing and Analyzing the Behavior of Feedback Systems
IV. Understanding Dynamic Problems
V. Introduction to Simulation
VI. Formulating and Analyzing Simulation Models

An overview of the content of each learning package follows.

I. Basic Concepts: Dynamic Problems, Systems, and Models

Embarking on a course of study in system dynamics, students need to understand the basic concepts that underlie the field. Three concepts are central; the focus is on dynamic problems; the intent is to consider the whole system of interacting parts from which a problem arises; and models are explicitly employed to carry out the analyses. Each of these concepts is explored briefly in learning package I to provide a foundation for the study of feedback systems which follows.

A problem is dynamic if it changes over time. Urban crime rates, for example, rise; the economy cycles, as do pendulums and people's feelings of depression and elation; central city populations decline, and so do reserves of natural resources.

A system is defined as a collection of parts operating together for a common purpose, but the concept is sometimes better left undefined to be inferred from examples. The notion should connote complexity, but it should also suggest a wholeness of perspective and the feeling that the whole is greater than the sum of its parts.

A model is a representation--usually a simplification--of some slice of reality. Pictures, verbal descriptions, graphs, sets of equations, and laws are models. Thinking can be characterized as the manipulation of mental models; the real system is never in one's head. Such a concept may seem too generalized to be useful, but realizing the central role models play in our thinking (especially in the scientific method) allows asking the proper question-not whether to develop a model, but what kind of model is most helpful for a given purpose.

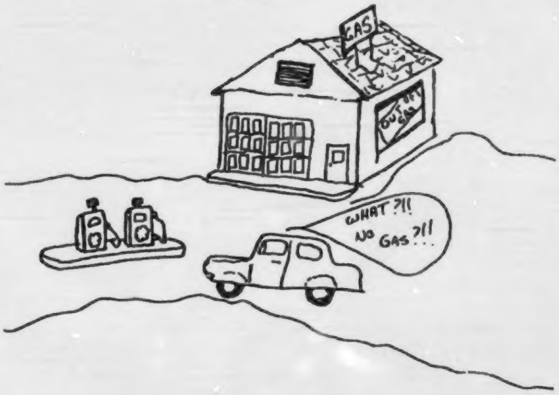
System dynamics helps us to understand problems arising in dynamic systems, first by making our mental models explicit, second by incorporating into them feedback relationships (learning packages II, III, and IV of the curriculum), and third by providing means for developing them into unambiguous mathematical models (learning packages V and VI) when the complexities become too great for mental models to handle.

II. Stucture of Feedback Systems

This part of the sequence introduces a way of thinking holistically about cause and effect. The principal tool employed to analyze a system is the causal-loop diagram. Cause-and-effect relationships are symbolized with arrows, forming chains of causal links. Loops result when some or all of these chains return to their starting points. These loops of causal influences are called feedback loops; they are the central focus and foundation of system dynamics.

Causal-loop diagrams are visual models of feedback systems. In this part of the curriculum they are used in a generally descriptive way to summarize the complex interactions in a variety of stories, problems, and systems with which the student is familiar. The following is one of the stories from this learning package followed by the possible causal-loop diagrams that students might develop.

THE OIL CRISIS


GAS
OUT OF GAS
WHAT ?!!
NO GAS ?!!...

One aspect of the oil crisis, as explained by an economist, was the starting of a vicious circle. This vicious circle was begun by agreements made by the Arab oil-producing countries in 1971 called the Teheran and Tripoli Agreements (named for the cities in which the meetings were held). Here these countries agreed to raise the price of oil. The rise in oil prices meant that these countries made more money. They made so much more money that they could not possibly spend it all. Realizing this, these countries decided not to produce as much oil. They knew that eventually their oil supplies would run out so they might as well make them last as long as possible.

Because there was less oil being produced in the world and more oil was needed every day, a scarcity developed. This scarcity of oil forced the oil prices to go up even higher, continuing the vicious circle.

The following causal-loop diagram explains the economist's vicious circle theory.

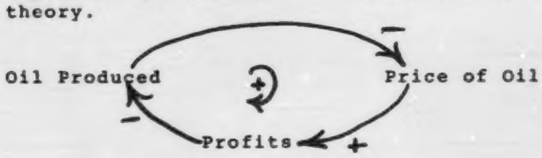


Figure 1. The Vicious Circle

In causal-loop diagramming, as in multiplication, two negative elements cancel each other out, creating a positive feedback effect.

The diagram below expands on the information given in this story and might result from a class discussion.

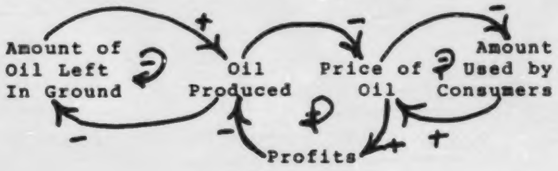


Figure 2. The Oil Crisis

The principal skills the students will develop in learning package II are the ability to represent the essential influences in a problem or system as a causal-loop model and the habit of searching for feedback influences which close causal loops. If students complete this part of the curriculum but do not continue on in the sequence, they will have gained a powerful tool for understanding complicated interactions, but they will have only begun to understand the connection between feedback loops and dynamic behavior.

III. Graphing and Analyzing the Behavior of Feedback Systems

Dynamic behavior of variables in systems is the focus of this learning package. The purpose is to understand how the structure of feedback loops is responsible for the behavior of variables in the system over time. Graphs are introduced and used intuitively, often without specifying numerical scales. The shape of the graph of a variable over time is the concern.

Behavioral characteristics of positive feedback loops and negative feedback loops are explored in simple situations and then exploited to analyze more complex multiple-loop structures. Positive loops are shown to be responsible for uncontrolled behavior such as exponential growth, while negative loops are shown to produce goal-seeking behavior, though often with disturbing fluctuations and cycles. S-shaped growth in several apparently different systems is shown to occur when a quantity is influenced first by a positive loop and then by a negative loop; a shift from dominance by the positive loop to dominance by the negative loop produces behavior which looks initially like unrestrained exponential growth but then becomes goal-seeking. An important principle will appear as different systems are explored: systems with the same feedback structure tend to behave the same over time, in the sense that the shapes of their graphs over time are essentially the same.

The role of delays in goal-seeking systems is explored intuitively. Students will become familiar with the principle that delays can cause a variable to oscillate around its goal.

The approach in this part of the curriculum is essentially non-quantitative, except that some quantitative graphing will be done to provide the tools required to graph intuitively the behavior in causal-loop diagrams.

Students completing learning package III will have a foundation for the principle that the behavior of a system is a consequence of its feedback structure. They will be ready to try to apply their understandings to more complex situations.

IV. Understanding Dynamic Problems

A number of recurring themes in real-world problems are uncovered in this part of the sequence. Each theme is explored in the context of several apparently different dynamic problems, making use of causal-loop models and the properties of feedback systems developed in Sections II and III. The structure of feedback loops responsible for the thematic behavior is exposed in each system, providing a common focus for understanding the different problems sharing that theme.

One of the themes explored here is sometimes referred to as the counterintuitive behavior of complex systems; well-intentioned policies often tend not to produce the behavior expected, occasionally producing results opposite to those intended. The phenomenon is traced initially to the distinction between open-loop and closed-loop thinking. The former overlooks feedback in contrast to the latter, which incorporates within the boundary of the system all the essential feedback influences. Counterintuitive behavior appears again as the significance of feedback structure is explored; feedback systems tend to resist certain kinds of change, unless the actual structure of the system is affected. In feedback systems the short-term effects of a policy may be different from, even opposite to, its long-term effects. The tendency of complex systems to become dependent upon external controls is explored under the theme of "shifting the burden to the intervener". The concept of tradeoffs is introduced. In complex systems, policies rarely improve all aspects of the system at once; usually a policy improves some areas and is deleterious to others, requiring policy-makers to take explicit account of the tradeoffs.

The problems explored in this part of the sequence range from peer pressure, cramming for a test, and mowing lawns, to criminal justice, pollution, urban growth-stagnation-decay, drug-related crime, and global population and food needs.

Learning package IV is the culmination of the essentially non-quantitative part of this introductory system dynamics curriculum. It shows the power of attempting to understand complex dynamic problems by focusing on feedback loops, to illustrate some themes which recur so frequently in complex systems that they may be called "principles", and to leave the students with a balanced view of the power and limitations of their understandings at this point. As learning packages II, III and IV have progressed, the need for knowing the relative strengths of feedback loops in a system will have arisen at various times, leading naturally (but not necessarily) to the next part of the sequence in which methods are developed for making the assumptions embodied in a causal-loop diagram unambiguous by quantifying them.

V. Introduction to Simulation

Computer simulation in system dynamics becomes necessary when the implications of a structure of feedback loops are in doubt. Greater precision is required than a causal-loop diagram can provide. Section V develops the skills needed to translate simple causal-loop models into quantitative models which a computer can trace through time, simulating the behavior of the actual system.

Two critical notions form the focus: the concepts of a level (or stock) and the concept of a rate (or flow). Level variables are pictured as rectangles; the rate adjusts the flow of something into the level with which it is associated, just as a faucet adjusts the flow of water into a tub, changing the water level.

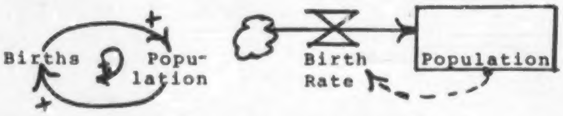


Figure 3. Causal-Loop Diagram as It Relates to a Flow Diagram

This part of the curriculum returns to the work of learning packages II and III, developing quantitative understanding of positive and negative feedback loops and the behavior of simple systems. Students will expend their skills in understanding and interpreting graphs of variables over time. In addition, they will develop abilities to write general level equations and the elementary rate equations for exponential growth and decay and sigmoid (S-shaped) growth. Exercises include first-hand calculating and graphing the model variables, then simulating the models with the aid of a computer. Familiarity with computers and programming is not required; the introduction to the simulation language DYNAMO is self-contained. The programming is presented as a means to an end: making assumptions sufficiently precise and suitably coded that a computer can trace out their implications over time. Having hand-simulated the models first, the students will know what the computer is doing.

At the close of learning package V the students are ready to understand more complex simulation models, they will have begun to see the power of simulation, and they will have solidified their understandings of the behavior of feedback loops covered intuitively in learning package II and III.

VI. Formulating and Analyzing Simulation Models

Several of the problems addressed in learning package IV are reconsidered, generally in greater detail. For each, in turn, a quantitative model is developed in DYNAMO, and explorations of the system are carried out by simulating different conditions in the model. The central focus, besides the significant problems themselves, is the understanding of complex system behavior. Where in a given system does intervention have the most effect? Why does the system behave as it does? What policies actually improve the behavior of the system? Why does one policy have a desirable effect while others which initially appear promising have little helpful effect or may even prove to be harmful?

Explorations of the behavior of a system are carried out, and alternative policies investigated by changing numerical relationships in the model, altering, or adding equations. The computer is shown as an obedient servant tracing out the implications of a modeler's assumptions over time. Each simulation model and each simulation run appear not as ends in themselves, but as means to understanding the dynamics of a certain problematic system. The goal is understanding, and feedback models help us to understand certain kinds of problems.

Learning package VI completes this introduction to system dynamics. Students continuing through all six sections will have a new understanding of the causes of dynamic problems and the beginnings of a set of tools for analyzing and understanding them. They will have learned to look at problems holistically and to search for feedback loops responsible for the behavior of the system. They will understand the role of models in approaching problems and what is required to develop such models. The students will also have an introduction to the meaningful role computer models and simulations can play in helping people to cope with the complex dynamic problems they face.

THE CURRENT CURRICULUM PROJECT

The project is scheduled for completion in August 1980. During the spring of 1980 it will be pilot tested in six high schools. Two teachers from each high school will integrate the curriculum into their courses. These teachers are from the disciplines of computer science, mathematics, chemistry, physics, English, history, environmental studies, and biology.

The targeted audience of secondary students need not be the only users of the materials. The materials that were created for fifth and sixth graders have been used in high schools, universities, and even in the MIT Sloan Fellows Executive Development Program. The secondary level design should also be appropriate for many undergraduate college classes, as well as for other possible learning settings. The learning packages have been planned to meet the following criteria:

1) Interdisciplinary and Unifying. The materials will provide examples from a number of subject areas: physics, economics, biology, ecology, sociology, anthropology, social studies, and literature. The generic similarity of structure and behavior in these fields will be illustrated.

2) Relevant. The materials will deal with significant world and national problems as well as problems encountered by students in their own lives.

3) Hands-on Approach. The set of learning packages will teach students the modeling process through a series of exercises. Initially, the students will develop on their own a set of feedback diagrams derived from the verbal description of a problem. Then they will go on to quantify the elements of the problem through graphing, and finally write the equations representing the structure of the model. The students will then hand simulate their model and finally simulate the model with the aid of a computer.

4) Supplementing Rather Than Replacing Existing Subjects. The curricular materials are designed to supplement rather than replace current courses. For example, the materials will be used to demonstrate integration in a mathematics course, population growth in a biology course, factors underlying social problems in a social science course.

5) Reiterative. Both the problems selected for study and the skills taught will reappear in several learning packages. The students therefore will study the same problem areas with increasingly more details of problem elaboration and increasingly more sophisticated methods as they proceed from causal-loop diagramming through graphing, to equation writing, hand-simulation, and then computer-simulation. Each skill that is taught will be recalled for use in later exercises that focus ostensibly on other skills development. Further, the student's sense of the similarity of underlying dynamic structures will also be increased as problems from a variety of disciplines are studied from a system's perspective. The students will begin to understand how identifying the underlying structure of a problem gives one a strong sense of comprehending the problem.

The curriculum development group hopes to do extensive field testing over the next few years. Should anyone be interested in field testing the learning packages or wish more information on the project, please contact either of the authors.

REFERENCES

1. Forrester, J.W. Industrial Dynamics. Cambridge, MA: MIT Press, 1956.
2. Odum, H.T. Energy, Power and Society. New York: Wiley-Interscience, 1971.
3. Roberts, N. "Teaching Dynamic Feedback Systems Thinking: An Elementary View." Management Science, Vol. 24, April 1978, pp. 836-843.

THE COMPUTER SOFTWARE TECHNICIAN PROGRAM AT PORTLAND COMMUNITY COLLEGE

1980

David M. Hata
Portland Community College
12000 S.W. 49th Avenue
Portland, Oregon 97219
(503) 244-6111

ABSTRACT

This paper is a status report on a new vocational program being offered at Portland Community College entitled "Computer Software Technician." The curriculum was developed jointly by Portland Community College and an advisory board composed of representatives from local electronics manufacturing companies such as Tektronix, Intel, and Electro-Scientific Industries and is targeted at software/hardware technician positions in the O.E.M. environment.

INTRODUCTION

Portland Community College is a comprehensive community college serving all of Washington County and parts of Multnomah,Columbia, Clackamas, and Yamhill counties, a region of some 679,000 people. Within the geographic boundary of the community college district are located several large manufacturers of electronic equipment -- Tektronix, Intel, E.S.I., and others.

Courses offered at Portland Community College are organized and integrated into a broad variety of programs ranging from associate degree, certificate career, and college transfer programs to special interest and enrichment courses, apprenticeship training, and high school completion. The philosophy of Portland Community College is to offer learning opportunities to everyone regardless of prior educational experiences, a philosophy which has earned the school the name "The Educational Shopping Center." It is in this environment that the Computer Software Technician Program is growing.

PROGRAM DEVELOPMENT

Identifying the need for computer software technicians began in late 1977. A survey conducted within various groups at Tektronix showed widely varying skills among those polled. For example, the Information Display Group's skill set was much more computer oriented, greater than 40 percent software, as opposed to less than 20 percent for the Test and Measurement Group. It was also noted that groups with more software engineers had fewer technicians. It became evident that the software equivalent of the hardware technician did not exist.

With the increase in the number of microcomputer-based systems being manufactured, shifts in the skill sets required in the manufacturing environment were projected to move toward increasing emphasis on software skills. Translated into future manpower requirements, an addition of several hundred software professionals was projected over the next five years, an impossible task in the face of a critical shortage of technical people. Using past experience in developing electronic technician programs at the two-year level, the two-year concept for training software technicians began to grow among managers.

Development of the program began in March 1978 when Tektronix formally approached Portland Community College with the idea of offering a two-year associate degree program to train computer software technicians. An ad hoc committee was formed and instructors from the Electronic Engineering Technology and Data Processing Departments were assigned to course development tasks.

PROGRAM GOAL

Existing programs in electronics and data processing/computer science emphasize either hardware-related or software-related topics. The goal of the Computer Software Technician Program is to train a technician who has the skills to write and develop microcomputer applications software under the guidance of a software engineer and can bridge the hardware/software gap by possessing skills that will enable him/her to verify correct system operation.

THE COMPUTER SOFTWARE CURRICULUM

The proposed Computer Software Technician Program consists of 101 credit hours for the associate of applied science degree. The first year consists of parallel hardware and software course sequences combined with mathematics and communications courses. Students develop basic problem-solving skills while building a foundation in basic circuit theory and device operation. The software and hardware areas merge in the fourth term as courses begin to integrate both software and hardware topics into a unified presentation. The program concludes with courses and on-the-job learning experiences, to create the ability to function as part of a development team and learn the art of project management.

		Cred. Hrs.	Lecture	Lab. Hrs.
	TERM I			
CST 2.211	Software Programming I	4	4	2
MTH 125	Computer Oriented Mathematics I	5	5	
WR 2.301	Business Communications I	3	3	
EL 6.117	Basic Electric Circuits	6	4	6
	TERM II			
CST 2.221	Software Programming II	4	4	2
MTH 126	Computer Oriented Mathematics II	5	5	
WR 2.302	Business Communications II	3	3	
EL 6.127	Fundamentals of Semiconductors	6	4	6
	TERM III			
CST 2.231	Software Programming III	4	4	2
CS 233	Intro. to Numerical Computation	4	4	2
WR 227	Technical Report Writing	3	3	
EL 6.137	Digital Logic Fundamentals I	6	4	6
	TERM IV			
EL 6.248	Introduction to Microprocessors	4	3	3
CST 2.241	Low Level Languages	4	4	2
SP 100	Basic Communication	3	3	
EL 6.247	Digital Logic Fundamentals II	4	3	3
	TERM V			
CST 2.123	Language Processors	4	4	2
CST 2.136	I/O & Data Communication Prog.	4	4	2
PSY 1.546	Psychology & Human Relations	3	3	
EL 6.257	Peripheral Circuits	4	3	3
	TERM VI			
CST 2.126	Project Management	4	4	2
CST 2.132	Operating Systems	4	4	2
CST 2.141	Field Project	6	1	18
EL 6.267	Advanced Micro Systems	4	3	3

IMPLEMENTATION

Laboratory Facilities:

Parallel efforts to upgrade existing electronics laboratory space used by the electronic engineering technology program and to acquire the specialized equipment necessary to implement the Computer Software Technician Program have produced a well equipped set of laboratories to support both programs. Our initial goal to equip each laboratory with industry-standard equipment has been realized. Specialized equipment for the Computer Software Program purchased for laboratory use includes logic analyzers, microcomputer trainers, and software development systems.

Equipment needed by laboratories:

Basic Circuits Laboratory:
- Dual-trace oscilloscope (35 MHz)
- Digital voltmeter

Basic Circuits Laboratory Continued:
- Triple power supply
- Function generator

Advanced Circuits Laboratory:
- Dual-trace oscilloscope (100 MHz)
- Digital voltmeter
- Triple power supply
- Function generator
- Digital counter

Digital Systems Laboratory:
- Dual-trace oscilloscope
- Digital voltmeter
- Function generator
- Triple power supply
- Logic analyzers
- Microcomputer trainers

Computational Systems Laboratory:

Microcomputer systems capable of running BASIC and PASCAL

Microcomputer development systems

Cooperative Work Experience:

A cooperative education program has been established with local companies, and ten CST students are currently participating in the program. Each cooperative education student is required to carry twelve credit hours of academic work while working twenty hours per week. Close supervision is maintained between the cooperative education coordinator and managers of the cooperative education students.

The response has been favorable among participating managers; student response has been positive although many have expressed the feeling that the academic work load combined with work assignments has been demanding.

Faculty Upgrading:

Program development has reinforced the need for a teaching faculty having a skill mix that spans the hardware/software boundary. To date, faculty upgrading has been accomplished by technical seminars, independent study, and interaction with industry. Keeping pace with an ever changing technology is a continual struggle, and incorporating new teaching ideas into courses remains before our faculty.

Course Content Guides:

Course content guides are being reviewed for content and updated. These course content guides are available upon request by writing to the author.

IMPACT ON COMPUTER EDUCATION

Data processing and computer science programs to date have been very business applications oriented. Curricula are heavily weighted toward programming languages and applications that do not fit the traditional product development cycle.

The Computer Software Program was a totally new program rather than a modification of an existing program. The electronic engineering technology and data processing programs were left intact since they are currently serving a well-defined industry need. In this manner, a curriculum was developed to meet a future need for technicians with both software and hardware skills.

With the first graduating class in June 1980, software technicians capable of fitting into systems development and manufacturing groups will be entering the job market. It has been evident from the feedback obtained through managers of our cooperative education students that the skill mix will be right for entry-level positions in software groups currently employing these students.

Programs of this type are opening up new areas in which persons interested in computers, computer programming, and electronics can enter. Programs of this type have not been available in the past. To industry, these technicians will provide the analogue of the hardware-oriented hardware technician that has been missing.

CONCLUSION

The need for computer software technician programs will grow, and the Computer Software Technician Program at Portland Community College can serve as a model for programs at other colleges. It is the result of close industry-college interaction and is currently being evaluated through a cooperative work program with local electronics companies.

A COMPUTER SCIENCE MAJOR IN A SMALL LIBERAL ARTS COLLEGE

Joerg Mayer
Department of Mathematical Sciences
Lebanon Valley College
Annville, PA 17003
(717) 867-1055

This presentation examines the difficulties, both practical and philosophical, which stand in the way of offering a computer science degree at a small four-year liberal arts college. Also, I will present the compromise which was struck at our college.

Until recently, computer science as an undergraduate degree program was almost entirely restricted to the larger universities. Such programs were expensive, both in terms of teaching personnel and equipment, and required an environment which included an engineering school. These conditions are easing somewhat, and it is becoming increasingly difficult for a small college to resist the incentives to enter the computer science market. Computer hardware is now within the means of most institutions, and for those with inadequate funding there are ties into a commercial or educational time-sharing network. Also, with declining overall enrollments on the horizon and an at best stable interest in most traditional subjects, the continuing shortage of college-trained computer personnel seems to offer the chance to shore up at least some of the drain in the student population by offering a major in computer science.

Encouraging though the emerging conditions may be, the arguments against adding computer science to the program of a liberal arts college are rather unsettling.

Despite the efforts of many, there remains a sizeable gap between the defenders of liberal education as the attempt to capture the essence of the complete man, and the proponents of an introduction to science and technology as the only means of preparing our youth for the technological future. The present mixture of major programs in most liberal arts institutions reflects more an uneasy truce than the result of mutual understanding. The separation between the two cultures shows no signs of dissolving, at least not in education.

Computer science as an undergraduate major brings this dichotomy in education into sharp focus, perhaps because this discipline is predominantly machine and process oriented. The aim of computer science is not to seek truth or to understand man and nature, and in that sense it is anti-humanistic. Computer scientists are, as seen by the humanist, essentially non-reflective, meaning that they do not concern themselves with the ethical and moral implications of their work. (Weizenbaum's book on that issue is not widely known; and, in any case, it does not seem to be very popular among his colleagues.) Therefore, computer science is considered by many to be antithetical to the basic philosophy of liberal education. "One only has to observe the nearest computer freak to find proof for the dehumanizing, indeed depersonalizing influence of that machine." Computer science, as a relatively young discipline--exuberant, irreverent, and in some ways irresponsible and antagonistic to deeply held cultural values--does not easily fit into the age-worn quilt of liberal education.

Even if the misgivings were somehow to be overcome, however, the practical problems of offering computer science in a small liberal arts college are most forbidding.

To begin with, financing the needed technical expertise is beyond the capabilities of most such institutions. It would be unwise, both politically and financially, for the college to hire a beginning Ph.D. in computer science for $25,000 when the average salary of the assistant professors in that college is $14,000 and that of full professors is $22,000. Another manpower problem arises from the ACM standards set in Curriculum '78. There it is recommended that "approximately six full-time equivalent faculty members are necessary. ..." With a student:faculty ratio of 14:1 this translates into roughly 80 computer science majors, a number not likely to be reached in an institution of 1500 students, which is the typical enrollment of a small four-year college.

Another difficulty lies in the courses which should be offered. According to Curriculum '78, a minimal program in computer science consists of twelve courses in computer science and seven courses in mathematics, making a total of 57 credit hours in the major. Rare is the liberal arts institution that would allow such a concentration of a student's time in his or her major.

Finally, there is the need for hardware and software. Again, Curriculum '78 established certain benchmarks. "It is essential that appropriate laboratory facilities be made available that are comparable to those in the physical . . . disciplines . . .. The initial budgetary support . . . may be substantial." Also mentioned are extensive software systems. The whole package, even if conservatively interpreted, presents a financial burden that reaches well into six-digit numbers.

Realistically, then, it is impossible for all but a handful of the best endowed small liberal arts colleges to find the total budgetary support for the start of a computer science major which approaches the standards set by Curriculum '78. So a student who wishes to major in computer science must limit his choice of a college to those with an enrollment of at least 5000. It means that the smaller college cannot enter the market of computer science education. Finally, it means that employers and graduate schools of computer science cannot reach into that rich pool of above-average, motivated, and broadly educated graduates.

At Lebanon Valley College we were drawn into the computer science field more by accident than by design. In the early seventies when it became clear that the major in actuarial science was very attractive to prospective students and could be so designed as not to violate the precepts of liberal education too severely, the Department of Mathematics decided to expand its offerings into the applicable areas. The department grew steadily, so that in 1978 there were 70 majors in a school whose total enrollment was 950. In 1975, enough computer science courses had been added to offer an informal concentration in computer science. The technical courses included programming (advanced), computer architecture and assembly language, data structures, and an independent study. Of the 36 mathematics graduates since 1977, six took computer-oriented jobs, seven became system analyst/programmers, and four entered highly respected computer science graduate departments.

Encouraged by these results, and strongly persuaded by the Admissions Office, a major in computer science was added to the programs of the Department of Mathematical Sciences. Its structure is as follows. All students in the Department, regardless of their major, take the same core curriculum which consists of:

Analysis	13 hours
Foundations of Mathematics	3 hours
Differential Equations	3 hours
Introduction to Computer Science	3 hours
Linear Algebra	3 hours

During the difficult sophomore year, the students assess both their capabilities and their strongest interest. Accordingly, some will drop out of computer science and others will shift into it. For that major the remaining requirements are:

Abstract Algebra	3 hours
Classical and Numerical Analysis	3 hours
Computer Organization & Assembler	3 hours
Data Structures	3 hours
Programming Languages & Compilers	3 hours
Internship	3 hours

The internship must be taken in some commercial or industrial computer operation, usually during the summer. The languages taught during the last three years are BASIC-PLUS, FORTRAN, and Assembler (PDP11/40). We strongly encourage the mastery of one additional language, such as RPG or COBOL.

There are other requirements. To give the students some background in the basic components of computer hardware, they must take a year of physics and work in a small computer science laboratory. Also required are six hours in psychology, and three hours in an ethics course designed to deal with the ethical issues inherent in modern technology, and computers in particular. Finally, having observed the often abominable technocratese of handbooks and manuals, we require that the computer science major take a three hour writing workshop.

Thus, the major consists of 31 hours in mathematics, including the Curriculum

'78 courses MA 1, MA 2, MA 5, MA 3, CS 17, CS 18, with MA 6 strongly encouraged; twelve hours in computer science, including CS 2, CS 3, parts of CS 4, also CS 7, and CS 8; 18 hours in what we consider related topics; and three hours internship. The totals are 46 hours in technical courses and 18 hours in supportive disciplines.

These requirements are in contrast to Curriculum '78, which sets a minimum technical requirement of 51 hours, of which 30 are in computer science.

The disadvantages of our approach are at least the following. It makes us uncomfortable to be in such disagreement with Curriculum '78, and we would not be prepared to adjust if the suggested program of Curriculum '78 were to develop from recommendations to quasi-accreditation requirements. Secondly, the rather narrow scope of the computer science courses must result in a limited appreciation of the broad sweep of that field. And finally, the substantial and difficult requirements in mathematics will most likely lead to a higher dropout rate than we are accustomed to because many students interested in computer science are mathematically not very gifted.

The advantages are that we will be able to increase the enrollment in our department and that all the students in the department are being exposed to the new discipline. For them, the main and overriding advantage is the flexibility which they enjoy during the first three years. To be able to switch, without loss of credit and required information, between mathematics, actuarial science, computer science (and probably operations research in the near future) is valuable to the mathematically gifted, who did not in their high school years have any indication of the many job and graduate school opportunities for which they may be predestined. For me personally, the greatest advantage of our approach is that it shows our students, and not just in theory, the great breadth of mathematics and the mutual dependence and influence of the various disciplines we incorporate in our total program.

Two conclusions appear to be clear. First, a small liberal arts college can offer a computer science major only if it does not adhere too closely to the recommendation of Curriculum '78. And second, such a major has any chance of success only if it is incorporated in the operations of an already existing department. Its faculty members must be willing to work hard mastering most of the courses in computer science while they are still teaching the normal twelve hour load. The Computer Center personnel must be sympathetic to the reality that the new major will tie up the system more often than they are accustomed to. And the administration must be willing to accept the fact that one cannot start a technology-based major without improving the existing hardware and hiring at least one specialist. It is well to remember that more often than not, the necessary support will come at least two years after it is needed.

For those who may be contemplating implementing a variation of our approach, let me close on a bright note. Within the first semester of the new major, four students transferred into it from other departments in the college, and two more students transferred into it from other schools because they wanted both computer science and a small liberal arts institution.

Author Index

Abernethy, Kenneth 179
Allard, Kim 230
Alpert, Elizabeth 96
Amann, G.W. 266
Anger, Frank D. 171, 249
Arenson, Michael 1

Baird, W.E. 266 - rec'd
Baldwin, R. Scott 46
Bass, George M. Jr. 38
Beck, James D. 245 - rec'd
Beidler, John A. 143
Bishop, Judy M. 147
Bork, Alfred 258
Bowman, William R. 16
Boyle, Thomas 214
Brown, Bobby 199
Brown, Guy Larry 256 - rec'd
Brown, Thomas 237
Burrows, Robert L. 220
Bush, Steve 19

Caldwell, Robert M. 31 - rec'd
Collins, Ronald W. 74
Corbet, Antoinette Tracy 54
Czejdo, Bogdan 112

Davis, A. Douglas 86
Deal, Ralph M. 281
DeKock, Arlan 138
Dempsey, Richard F. 152
Dershem, Herbert L. 65 - rec'd
Dorn, William 75

Effarah, Jamil E. 25
Ellison, Robert J. 68

Franklin, Stephen 258
Friedman, Frank L. 103
Fuhs, F. Paul 205

Garraway, Hugh 200
Garson, James W. 42
Gatz, A. John, Jr. 76
Gordon, Sheldon P. 169
Gruener, William 198
Groom-Thornton, Joan C. 54

Hagee, Michael W. 90
Haiduk, H. P. 222
Hannay, David G. 119
Hata, David M. 287
Hausmann, Kevin 2, 170
Hazen, Paul 73
Hopper, Judith A. 62
Hunter, Beverly 168
Hybl Albert 276 rec'd

Johnson, Dale M. 46
Jordan, Eleanor W. 7, 260 - rec'd

Kneifel, David 37
Kurtz, Barry 258

LaFrance, Jacques E. 261
Little, Joyce Currie 220
Lim, Tian S. 90
Lovis, Frank 75

McVay, Peter O. 271 rec'd
Magnant, Peter 214
Maner, Walter 221
Marshall, Sr. Patricia 238
Mayer, Joerg 290
Meinke, John G. 143
Mebane, Donna Davis 197
Mebane, Rodney 197
Mocciola, Michael 236
Morgan, Catherine E. 168
Moshell, J.M. 266 - rec'd
Moursund, David 125

Oliver, Lawrence 137, 169
Olivo, Richard F. 81 - letter rec'd

Piegari, G. 179
Poirot, James L. 130
Pollak, Richard A. 90
Powell, James D. 130

Rahmlow, Harold 1
Reggia, James A. 276 - rec'd
Roberts, Nancy 281
Rodriguez, Rita Virginia 171

Schimming, Bruce B. 58
Shelly, Gary 3
Sobol, Thomas 155 rec'd
Stokes, Gary 4
Stoutemyer, David R. 194 - rec'd
Sustik, Joan 199

Taylor, Robert P. 130, 155
Taylor, Timothy 223
Thorkildsen, Ron 230
Thorsen, A.L. 179
Tinker, Robert F. 250
Turner, A.J. 6

Varanasi, Murali R. 5

Ward, Darrell L. 19
Wehrle, Howard F., III 12
Wetmore, David E. 139 - rec'd
Whittle, John T. 65 rec'd
Wong, Pui-Kei 184 - rec'd